Circumventing Server Bottlenecks: Indirect Large-Scale P2P Data Collection

Di Niu, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto
{*dniu, bli*}*@eecg.toronto.edu*

Abstract

In most large-scale peer-to-peer (P2P) applications, it is necessary to collect vital statistics data — sometimes referred to as logs *— from up to millions of peers. Traditional solutions involve sending large volumes of such data to centralized logging servers, which are not scalable. In addition, they may not be able to retrieve statistics data from departed peers in dynamic peer-to-peer systems. In this paper, we solve this dilemma through an indirect collection mechanism that distributes data using random network coding across the network, from which servers proactively* pull *such statistics. By buffering data in a decentralized fashion with only a small portion of peer resources, we show that our new mechanism provides a "buffering" zone and a "smoothing" factor to collect large volumes of statistics, with appropriate resilience to peer dynamics and scalability to a large peer population.*

1 Introduction

Collecting vital statistics data from peers has become critical to monitor and diagnose large-scale peer-to-peer (P2P) multimedia applications, especially when they are in live operation as a commercial platform. Traditionally, these commercial P2P live streaming systems use *logging servers* to collect vital statistics data from the participating peers periodically. Such statistics data consist of measurements of important performance metrics in the P2P application at each peer, and are used by network administrators and analysts to improve the protocol design or to troubleshoot network outage.

It has recently become apparent that the use of centralized logging servers is not *scalable* to the large number of participating peers. As has been observed in previous P2P measurement studies such as measurements from UUSee Inc. [14, 15] — a leading provider in mainland China for P2P live streaming solutions — periodic statistics data collection actually consumes a very substantial volume of traffic, especially when the number of peers in the session increases dramatically in a short period of time. Such periodic reporting essentially morphs into a *de facto* Distributed Denial of Service (DDoS) attack to the logging servers, as the server bandwidth is not sufficient to handle an excessive number of simultaneous uploading flows, with either TCP or UDP as the transport layer.

Naively, these logging servers may be able to mitigate this problem by periodically changing the peers that they proactively pull data from. However, with limited bandwidth, the servers can only download from a small subset of all the peers at the same time, leaving most of the peers waiting for service. Aggravated by the phenomenon of peer dynamics, a large number of peers may have already left the system before any of the servers actually become available to "pull" from them, losing such statistics permanently. Ironically, since peers tend to leave soon after the quality degrades, such statistics from departed peers may be the most useful to diagnose system outages or protocol deficiencies! Even if all peers may eventually be probed by logging servers, the bandwidth available on the pair-wide Internet path between a peer and a central logging server may be too low to successfully report all vital statistics.

In a nutshell, with current solutions using centralized logging servers, the amount of data that can be instantaneously collected from a large-scale P2P system is strictly limited by the server bandwidth. While such bandwidth may be more than sufficient to handle the *average* rate of collecting such vital statistics, it may not be able to accommodate flash crowds of peer arrivals and large-scale peer departures in a short period of time. In order to achieve better scalability, one possible remedy is to increase the number or bandwidth of such logging servers, trying to accommodate the *peak* load of collecting statistics, rather than the *average*. This is extremely costly and fails to take advantage of statistical multiplexing of resources. Since the task of statistics collection is usually not time-sensitive, we are able to trade-off *timeliness* for bandwidth, by *allowing peers to buffer such data in a decentralized fashion*, much like a distributed storage for vital statistics.

In this paper, we design, analyze, and simulate an indirect data collection mechanism that is specifically designed for collecting vital statistics in large-scale peer-to-peer systems, such as P2P streaming. Our design has two simple objectives: *scalability*, where the design must be scalable to handle a large number of peers, with some trade-offs on delay; and *resilience*, where losses of vital statistics from peers that have recently departed should be kept to the minimum.

In particular, our new mechanism requires that peers first exchange their statistics data blocks with their neighbors using random network coding and probabilistic gossip protocols. By utilizing peer resources to form a "buffering" pool, the transmission of statistics from peers to logging servers is "cushioned," such that server bandwidth is provisioned to handle only *average* load, rather than the *peak*. To some extent, our new mechanism is akin to delay tolerant networks: by spreading data blocks across the network, the data that the servers are unable to collect immediately are stored for future delivery in a delayed fashion. Such "cushioning" also plays an important role to prevent losing vital statistics of departing peers.

The remainder of this paper is organized as follows. In Sec. 2, we describe our algorithms for indirect data collection in P2P networks. In Sec. 3, we interpret network events via a random bipartite graph process and formulate a system of ordinary differential equations (ODEs) to characterize system states. Based on these ODEs, we evaluate the performance of our new scheme in Sec. 4. Simulation results are presented along with the analytical results. Finally, we discuss related work in Sec. 5, and conclude the paper in Sec. 6.

2 Algorithm Description

We consider the problem of collecting vital statistics data from a total number of N peers in a P2P network. To accommodate the fluctuating nature of the upload demand at each peer, we assume original statistics data blocks are generated at each peer in a Poisson process with rate λ so that the average upload demand of each peer is λ. Each peer has an average upload bandwidth of μ set aside for reporting statistics.

Fig. 1(a) illustrates the traditional mechanism of using logging servers to "pull" directly from peers. Such a scheme is not scalable, as the aggregate bandwidth of logging servers may not be able to accommodate the peak load of reporting vital statistics. Furthermore, if any peer quit the session before it can be serviced by any of the servers, the statistics in it will be lost permanently. In the case of collecting vital statistics for postmortem analysis, we are most likely able to trade-off the timeliness of retrieving such data, and tolerate some degree of delay. Thus, in our approach, we require peers exchange data blocks with each other periodically by a gossiping protocol that will be described later. Servers can retrieve the desired data by randomly probing the online peers, shown in Fig. 1(b). This approach not only eases the server bottleneck by buffering up data for future delivery, but also achieves better data persistence in the event of peer dynamics.

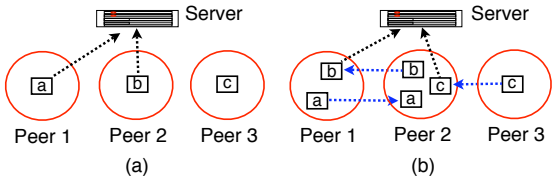

Figure 1. (a) Traditional pull-based approach (b) Indirect data collection.

In order to further increase data persistence, we leverage the wisdom of random network coding [6], [1], [5] to spread data blocks across the network in their coded form. As coded blocks are all equally useful, network coding eliminates the risk suffered by the non-coding case, where certain data blocks could become rare and thus more vulnerable to losses than other blocks upon peer departures [8]. However, the computational complexity of network coding mandates that, in reality, network coding needs to be performed within *segments*. The basic idea of *segment based network coding* or *group network coding* [1] is to group the original data blocks produced at each peer into *segments*, each with a prescribed number of s blocks, a quantity we refer to as the *segment size*. A random linear code (RLC) is applied to each segment in the following way:

Assume a certain segment i generated at peer A has original blocks $\mathbf{B^{(i)}} = [B_1^i, B_2^i, \ldots, B_s^i]$, then a coded block b from segment i is a linear combination of $[B_1^i, B_2^i, \ldots, B_s^i]$ in the Galois field GF(2^8). Coding operation is not limited to the source: if a peer (including the source) has buffered l ($l \leq s$) coded blocks of segment i $[b_1^i, b_2^i, \ldots, b_l^i]$, when transferring to another peer p, it independently and randomly chooses a set of coding coefficients $[c_1^p, c_2^p, \ldots, c_l^p]$ in GF(2^8), and encodes all the blocks it has from segment i, and then produces one coded block $x = \sum_{j=1}^{l} c_j^p \cdot b_j^i$. The coding coefficients used to encode *original blocks* to x are embedded in the header of the coded block. As soon as the server has collected a total of s coded blocks from segment i $\mathbf{x} = [x_1^i, x_2^i, \ldots, x_s^i]$ that are linearly independent, it will be able to decode segment i. The decoding complexity turns out to be approximately $O(s)$ operations per input block [8]. Naturally, we can vary the coding complexity by changing the segment size. When network coding is applied in segments of size s, segments of s blocks are injected at each peer in a Poisson process with rate λ/s, resulting in a same block rate of λ. Note that $s = 1$ indicates the non-coding case.

Coded blocks are spread across the network by the following gossiping protocol: 1) At rate μ, each peer, say peer A, chooses a segment r uniformly at random (u.a.r.) from among all the segments of which it has at least one (coded) block in its buffer to generate a coded block q; 2) A then transmits q to peer B chosen u.a.r. from among its neighbors which have not received s linearly-independent coded blocks of segment r. (The neighbors of a peer are the peers that maintain data connections with it.)

To ensure smooth data collection in light of fluctuating traffic and peer dynamics, the servers adopt a coupon-collector-like algorithm to collect data. Assume there are N_s servers in collaboration, each with a capacity of c_s. For the simplicity of notation in the analysis, we define the *normalized server capacity* as $c := c_s N_s/N$. At rate c_s, each server chooses a peer p u.a.r. from among all the peers with non-null buffers and chooses a random segment in peer p, which then transmits one coded block of this segment to the server. Note that servers may collect redundant blocks of a segment that is already decodable, as no buffer comparison is made between a server and peers or among the servers. However, we will show that even with such simplicity, the collection efficiency, or the session throughput can be made close to optimal probabilistically by adjusting the segment size for coding and various other parameters.

To prevent redundant data from flooding the network, we require each (coded) block to have a time to live (TTL) at each peer exponentially distributed with mean $1/\gamma$. A (coded) block is deleted once its TTL has expired to make storage spaces for newly generated or obtained blocks. Moreover, each peer's buffer is set to have a cap of size B. If a peer's buffer is full, it will not accept blocks from its neighbors.

3 A Differential Equation Characterization

In what follows, we map the proposed algorithms for indirect data collection onto a random bipartite graph process, which can be asymptotically characterized by a set of ODEs as the number of peers $N \to \infty$. As illustrated in Fig. 2, segments and peers correspond to vertices on different sides of a bipartite graph G respectively. For every (coded) block of segment r in peer A, there is an edge in G between peer A and segment r. Multiple edges can be allowed for a segment-peer pair, as each peer can buffer up to s linearly independent (coded) blocks of a certain segment. For example, in Fig. 2, Peer 1 has three blocks from Segment 1 and one block from Segment 3. Servers are not shown in G.

Denote by $X_i(t)$ the number of segments of degree i at time t and by $Y_i(t)$ the number of peers of degree i. A segment is of degree i if and only if there are i blocks of this segment in the network. A peer is of degree i if and only if it contains i blocks in its buffer. The total number of blocks in the network equals to the total number of edges in G, denoted by $E(t)$. Let $z_i := \frac{1}{N} Y_i$ and $w_i := \frac{1}{N} X_i$ denote the rescaled degree sequences and $e(t) := \frac{1}{N} E(t)$ be the average number of blocks in each peer. Actions of the peers and servers can be interpreted by the following graph operations:

- Segment Injection (Adding Edges and Vertices). At rate λ/s, s new edges are added to each peer in G whose degree is no more than $B - s$, together with a new segment incident to these s edges.
- Block Encoding and Transfer (Adding Edges). At rate μ, each peer s in G whose degree is non-null picks u.a.r. a segment p from all the segments adjacent to it and picks u.a.r. from all the peers a peer d that needs blocks of segment p, and whose degree is less than B. Add a new edge pd to G.
- Block Deletion (Deleting Edges). At rate γ, each edge in G is deleted. Particularly, if a segment in G has degree zero after an edge is deleted, then the segment is deleted from G.
- Server Collection. Assume each segment is in one of the $s+1$ states; a segment is in state i if and only if the servers have collected i linearly independent blocks of this segment ($i = 0, 1, \ldots, s$). At rate $c_s = c \cdot N/N_s$, each server chooses a peer d u.a.r. from all the peers with non-zero degrees. It then chooses a segment p u.a.r. from all the segments adjacent to peer d. Increase the state value of segment p by one if its state value is less than s. Otherwise, do not increase its state value, since a segment in state s can already be decoded and any further collection of blocks from this segment will be redundant.

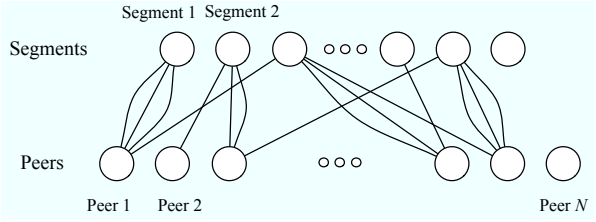

Figure 2. A bipartite graph representation.

We now formulate a system of ODEs on the graph process G_t mentioned above to characterize its asymptotic behavior as $N \to \infty$. We take the approach of considering the expected changes of X_i and Y_i contributed by different operations as G_t evolves to establish the system behavior in its limiting case. The proof of the correctness of this approach and the asymptotic approximation of the derived ODEs to the underlying process G_t can be found in [12].

Let us first consider block encoding and transfer. Consider the change of Y_i in Δt, during which only one action of edge addition or deletion occurs. When a segment p is

chosen for encoding, a peer d is chosen u.a.r. from all the peers with degrees less than B that still need blocks of segment p. Edge pd is then added. The probability of choosing a degree i peer is $(Y_i(t) - F_{pi}(t))/(N - Y_B(t) - F_p(t))$, where $F_{pi}(t)$ denotes the number of edges already present between segment p and peers of degree i, and $F_p(t)$ denotes the number of edges present between p and all the peers, which equals to the degree of p. As the maximum degree of any peer is bounded by B, it turns out that $F_{pi}(t)$ and $F_p(t)$ can only take finite values as well, that is $F_{pi}(t) \leq F_p(t) = o(N)$ for all p. Thus, the expected number of peers of degree i changing to $i+1$ in Δt is

$$\begin{aligned} & 1 \cdot \frac{Y_i(t) - o(N)}{N - Y_B(t) - o(N))} \cdot (N - Y_0(t))\mu\Delta t \\ = & \left(\frac{Y_i(t)}{N - Y_B(t)} + o(1)\right) \cdot (N - Y_0(t))\mu\Delta t, \end{aligned}$$

where $(N - Y_0(t))\mu\Delta t$ is the probability that one edge is added in Δt. When a peer changes its degree from i to $i+1$, Y_i decreases by one and Y_{i+1} increases by one. Hence,

$$\begin{aligned} & \boldsymbol{E}\{Y_i(t+\Delta t) - Y_i(t)|G_t\} \\ = & \left(\frac{(1-\delta_{i0})Y_{i-1}(t) - (1-\delta_{iB})Y_i(t)}{N - Y_B(t)} + o(1)\right) \\ & \cdot (N - Y_0(t))\mu\Delta t, \end{aligned}$$

where $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0$ if $i \neq j$ is the Kronecker delta function. Recall that $z_i(t) := \frac{1}{N}Y_i(t)$. Dividing by $N \cdot \Delta t$ on both sides, we get

$$z_i'(t) = \frac{(1-\delta_{i0})z_{i-1}(t) - (1-\delta_{iB})z_i(t)}{1 - z_B(t)} \cdot (1 - z_0(t))\mu, \quad (1)$$

To determine the change of X_i due to segment encoding and transfer, we assume the followings are equivalent: 1) Choosing a peer s u.a.r. from all the peers with non-zero degrees and then choose a segment p u.a.r. from all the segments adjacent to peer s in G_t 2) Choosing a segment p from all the segments with probability $\deg(p)/E(t)$, where $\deg(p)$ is the degree of segment p and $E(t)$ is the total number of edges in G_t.

This essentially means that the more blocks a segment has in the network, the higher the chance it is chosen by the servers. Thus, when adding an edge due to segment encoding and transfer, the probability of a degree i segment being chosen is $iX_i(t)/E(t)$. And the expected number of segments of degree i changing to degree $i+1$ in Δt is $(iX_i(t)/E(t)) \cdot (N - Y_0(t))\mu\Delta t$. Similarly,

$$\begin{aligned} & \boldsymbol{E}\{X_i(t+\Delta t) - X_i(t)|G_t\} \\ = & \frac{(i-1)X_{i-1}(t) - iX_i(t)}{E(t)} \cdot (N - Y_0(t))\mu\Delta t, \end{aligned}$$

Dividing by $N\Delta t$ on both sides, we get

$$w_i'(t) = \frac{(i-1)w_{i-1}(t) - iw_i(t)}{e(t)} \cdot (1 - z_0(t))\mu. \quad (2)$$

Let us now consider block deletion. In Δt, an edge is deleted with probability $E(t)\gamma\Delta t$ from G_t. Since the deletion process is Poisson, this edge is uniformly distributed among all edges. The probability that the deleted edge is adjacent to a degree i peer is $iY_i(t)/E(t)$. By a similar argument, we get the evolution of z_i under block deletion:

$$\begin{aligned} z_i'(t) = & ((1-\delta_{iB})(i+1)z_{i+1}(t) - iz_i(t)) \cdot \gamma, \\ & i = 0, 1, \ldots, B. \end{aligned} \quad (3)$$

Similarly, the evolution of w_i is given by

$$w_i'(t) = ((i+1)w_{i+1}(t) - iw_i(t)) \cdot \gamma, \quad i = 1, 2, \ldots \quad (4)$$

Finally, we consider the contribution of segment injection. A segment is injected into the network in Δt with probability $(N - Y_{(f)}(t)) \cdot \frac{\lambda}{s} \cdot \Delta t$, where $Y_{(f)}(t) = \sum_{k=B-s+1}^{B} Y_k(t)$. This segment appears in a degree-i peer with probability $Y_i(t)/(N - Y_{(f)}(t))$. Thus, the expected number of degree-i peers changing to degree $i+s$ is simply $[(N - Y_{(f)}(t))\lambda\Delta t/s] \cdot [Y_i(t)/(N - Y_{(f)}(t))] = Y_i(t)\lambda\Delta t/s$. Hence, when B is large enough, we have

$$\begin{aligned} & \boldsymbol{E}\{Y_i(t+\Delta t) - Y_i(t)|G_t\} \\ = & \begin{cases} -Y_i(t)\lambda\Delta t/s & i = 0, \ldots, s-1 \\ (Y_{i-s}(t) - Y_i(t))\lambda\Delta t/s & i = s, s+1, \ldots \end{cases} \end{aligned}$$

Dividing by $N \cdot \Delta t$ on both sides, we get

$$z_i'(t) = \begin{cases} -z_i(t) \cdot \frac{\lambda}{s} & i = 0, 1, \ldots, s-1 \\ (z_{i-s}(t) - z_i(t)) \cdot \frac{\lambda}{s} & s, s+1, \ldots \end{cases} \quad (5)$$

As for the change of X_i, a new segment with degree s is added whenever a segment is injected. Thus, the expected block increase is $(N - Y_{(f)}(t)) \cdot \frac{\lambda}{s} \cdot \Delta t$ for degree-s segments and zero for other segments. When B is large, we have

$$w_i'(t) = \delta_{is} \cdot \frac{\lambda}{s} \cdot (1 - z_{(f)}(t)) = \delta_{is} \cdot \frac{\lambda}{s}, \quad i = 1, 2, \ldots. \quad (6)$$

Putting (1), (3), and (5) together, and (2), (4), and (6) together, we finally arrive at the following ODE systems ($z_{-1}(t) \equiv 0$):

$$\begin{aligned} z_i' = & \frac{z_{i-1} - z_i}{1 - z_B} \cdot (1 - z_0)\mu - z_i \cdot \frac{\lambda}{s} + ((i+1)z_{i+1} - iz_i)\gamma, \\ & i = 0, 1, \ldots, s-1, \\ z_i' = & \frac{z_{i-1} - z_i}{1 - z_B} \cdot (1 - z_0)\mu + (z_{i-s} - z_i) \cdot \frac{\lambda}{s} \\ & + ((i+1)z_{i+1} - iz_i)\gamma, \quad i = s, s+1, \ldots, \end{aligned} \quad (7)$$

$$w_i' = \frac{(i-1)w_{i-1} - iw_i}{e(t)} \cdot (1-z_0)\mu + ((i+1)w_{i+1} - iw_i)\gamma + \delta_{is} \cdot \frac{\lambda}{s}, \quad i = 1, 2, 3, \ldots. \tag{8}$$

To characterize the server collection process, we introduce the *segment collection matrix* $\boldsymbol{M} = (M_i^j(t))$, $i = 1, 2, \ldots$, $j = 0, 1, \ldots, s$, where $M_i^j(t)$ denotes the number of degree-i segments that have j (coded) blocks collected by the servers already. Let $\boldsymbol{m} = (m_i^j(t)) := (\frac{1}{N} M_i^j(t))$ be the *rescaled segment collection matrix*. If we only consider the effects of segment encoding, transfer, and block deletion, the behavior of $m_i^j(t)$ is similar to that of $w_i(t)$:

$$\frac{\mathrm{d}m_i^j}{\mathrm{d}t} = \frac{(i-1)m_{i-1}^j - im_i^j}{e(t)} \cdot (1-z_0)\mu + ((i+1)m_{i+1}^j - im_i^j)\gamma, \tag{9}$$

Consider server collection operation. By the linear approximation mentioned above, each block is collected by the servers with a probability proportional to its degree in G_t. Since M_i^j increases by one and M_i^{j-1} decreases by one whenever a server obtains a block from a degree-i segment which already has $(j-1)$ blocks collected by the servers, the expected change of $M_i^j(t)$ is

$$\boldsymbol{E}\{M_i(t+\Delta t) - M_i(t) | G_t\} = \frac{i(M_i^{j-1}(t) - M_i^j(t))}{E(t)} \cdot cN\Delta t \tag{10}$$

Dividing by $N \cdot \Delta t$ on both sides, including the cases for $j = 0, s$, and also considering the effect of segment injection, we get for $i = 1, 2, \ldots$

$$\frac{\mathrm{d}m_i^j(t)}{\mathrm{d}t} = \begin{cases} -c \cdot im_i^0(t)/e(t) + \delta_{is}\lambda/s & j = 0, \\ c \cdot i(m_i^{j-1}(t) - m_i^j(t))/e(t) & j = 1, \ldots, s-1, \\ c \cdot im_i^{s-1}(t)/e(t), & j = s. \end{cases} \tag{11}$$

Combining all the effects in (9) and (11) together, we have for $i = 1, 2, \ldots$:

$$\begin{aligned} \frac{\mathrm{d}m_i^0(t)}{\mathrm{d}t} &= \frac{(i-1)m_{i-1}^0 - im_i^0}{e(t)} \cdot (1-z_0(t))\mu \\ &\quad + ((i+1)m_{i+1}^0 - im_i^0) \cdot \gamma - \frac{im_i^0(t)}{e(t)} \cdot c + \delta_{is} \cdot \frac{\lambda}{s}, \\ \frac{\mathrm{d}m_i^j(t)}{\mathrm{d}t} &= \frac{(i-1)m_{i-1}^j - im_i^j}{e(t)} \cdot (1-z_0(t))\mu \\ &\quad + ((i+1)m_{i+1}^j - im_i^j) \cdot \gamma + \frac{i(m_i^{j-1} - m_i^j(t))}{e(t)} \cdot c, \\ &\quad j = 1, 2, \ldots, s-1 \\ \frac{\mathrm{d}m_i^s(t)}{\mathrm{d}t} &= \frac{(i-1)m_{i-1}^s - im_i^s}{e(t)} \cdot (1-z_0(t))\mu \\ &\quad + ((i+1)m_{i+1}^s - im_i^s) \cdot \gamma + \frac{im_i^{s-1}}{e(t)} \cdot c. \end{aligned} \tag{12}$$

Our analytical results will be drawn from (7), (8), (12), which characterize system evolution.

4 Performance Evaluation

In this section, we evaluate the proposed protocol in terms of storage overhead, session throughput, data delivery delay, and loss resilience. We derive analytical results with respect to these metrics from the differential equations and provide more insights with simulations. Based on the analytical and simulation results in this section, we discuss how to choose parameters for the proposed protocol in various settings.

Let us consider the steady-state network by setting the derivatives in (7), (8), (12) to zero. Let $\tilde{z_i}(t)$, $\tilde{w_i}(t)$, $\tilde{m_i^j}(t)$ and $\tilde{e}(t)$ denote the steady-state value for $z_i(t)$, $w_i(t)$, $m_i^j(t)$ and $e(t)$. First, we can show the storage overhead of the indirect collection protocol is upper-bounded by μ/γ in the following theorem.

Theorem 1 *(Storage Overhead) Assume the buffer size B is large enough, regardless of the value of s, in steady-state network, the average number of (coded) blocks in a peer's buffer is*

$$\rho = (1 - \tilde{z}_0)\mu/\gamma + \lambda/\gamma, \tag{13}$$

Moreover, the average storage overhead of a peer is

$$\textit{Overhead} = (1 - \tilde{z}_0)\mu/\gamma < \mu/\gamma. \tag{14}$$

where $\tilde{z}_0$ is given by $\tilde{z}_0 = e^{-(1-\tilde{z}_0)\mu/\gamma - \lambda/\gamma}$ for $s = 1$, and is given by the steady-state solution to (7) for $s \geq 2$.

Proof: Setting the derivatives in (7) to zeros, after tedious yet straightforward deduction, we can get $\tilde{z}_i = \tilde{z}_0\rho^i/i!$, $i = 0, 1, \ldots, B$, where $\rho = (1-\tilde{z}_0)\mu/(1-\tilde{z}_B)\gamma + \lambda/\gamma$. And $\tilde{z}_0$ can be determined through $\sum_{i=0}^{B} \tilde{z}_i = 1$. If buffer size B is large enough, we have $1 = \sum_{i=0}^{B} \tilde{z}_i = \tilde{z}_0 \cdot \sum_{i=0}^{B} \rho^i/i! \approx \tilde{z}_0 \cdot e^{\rho}$. Thus, $\tilde{z}_0 \approx e^{-\rho}$. Since $\tilde{z}_B \approx 0$ when B is large, we have $\rho = (1-\tilde{z}_0)\mu/\gamma + \lambda/\gamma$.

Take the summation of all $\tilde{z}_i$, we get the rescaled total number of edges in the steady state as $\tilde{e}(t) = \sum_{i=0}^{B} i\tilde{z}_i = \rho \cdot \sum_{i-1=0}^{B-1} \tilde{z}_0\rho^{i-1}/(i-1)! = \sum_{j=0}^{B-1} \tilde{z}_j \cdot \rho = (1-\tilde{z}_B)\rho \approx \rho$, when B is large. Since the number of edges in G equals to the total number of block copies in the network, we have the average number of block copies at each peer equals to $\frac{1}{N} \cdot \sum_{i=0}^{B} i\tilde{Y}_i = \sum_{i=0}^{B} i \cdot \frac{\tilde{Y}_i}{N} = \sum_{i=0}^{B} i \cdot \tilde{z}_i = \tilde{e}(t) = \rho$. □

Theorem 1 shows the storage overhead of the proposed protocol can be limited to a relatively small value by setting the bandwidth μ and block deletion rate γ appropriately. In our simulations, μ/γ is set to be less than 20.

We define the session throughput as the actual rate (blocks/unit time) at which servers obtain original data. We can show that if the fluctuating traffic is modeled as Poisson processes at the peers, with the proposed indirect collection mechanism, servers provisioned with bandwidth that only handles an average traffic can achieve a throughput close to the throughput capacity for the session.

Theorem 2 *(Session Throughput) Assume $c < \mu$. The* **throughput capacity** *of the session is the aggregate server capacity: $C = c_s \cdot N_s = c \cdot N$. For the non-coding case ($s = 1$), in steady-state network, the session throughput is given by*

$$\textit{Throughput}(1) = N\lambda \cdot (1 - \frac{1}{\theta_+}), \qquad (15)$$

where θ_+ is the maximum root of $\alpha_2 x^2 + \alpha_1 x + \alpha_0 = 0$, $\alpha_0 = -q\gamma$, $\alpha_1 = q\gamma + \gamma + \frac{c}{\rho}$, $\alpha_2 = -\gamma$, and $q = 1 - \lambda/\rho\gamma$. Whereas for the coded case with a segment size s ($s \geq 2$),

$$\textit{Throughput}(s) = Nc \cdot (1 - \sum_{i=1}^{\infty} i\tilde{m}_i^s(t)/\rho), \qquad (16)$$

where $\tilde{m}_i^s(t)$ is the steady-state solution to (12).

Proof: The collection efficiency of servers η in the long run equals to the probability that a server collects a block from a segment needed by the servers in each trial in the steady state, which equals to the probability that a server picks a segment whose blocks have been pulled for less than s times. Thus, we have $\eta = 1 - \sum_{i=1}^{\infty} i\tilde{m}_i^s(t)/\tilde{e}(t)$. Thus, the session throughput is given by $C \cdot \eta = Nc\eta$. When $s = 1$, we can explicitly solve for $\tilde{m}_i^s(t)$ and $\tilde{e}(t)$ to obtain $\eta(1) = \frac{\lambda}{c}(1 - \frac{1}{\theta_+})$ and (15). □

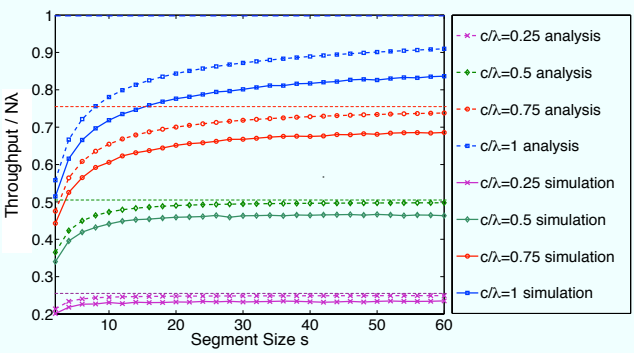

Figure 3. Session throughput as a function of segment size s. $\lambda = 20$, $\mu = 10$, $\gamma = 1$.

We plot the numerical results regarding throughput according to Theorem 2 as well as the simulation results in Fig. 3. In the Y-axis, we normalize the session throughput by dividing it by $N \cdot \lambda$, that is the aggregate peer upload demand. Each dashed horizontal line in the figure denotes the throughput capacity for a certain value of c.

It is clearly shown in Fig. 3 that by increasing segment size s, session throughput can be made close to the throughput capacity under that value of c. This is because when network coding is used, all coded blocks from a segment are equally useful, and the probability that servers collect redundant blocks using the coupon-collector algorithm falls down. This is the essential insight behind Theorem 2. Moreover, the use of a small segment size (*e.g.* around 20∼30) is sufficient to achieve high throughput as shown in the figure, with an acceptable computational complexity incurred. It is also worth noting that it is harder for the throughput to approach its capacity (dashed lines) as the normalized server capacity c increases, since the benefit of an indirect mechanism is more salient when server capacity is insufficient.

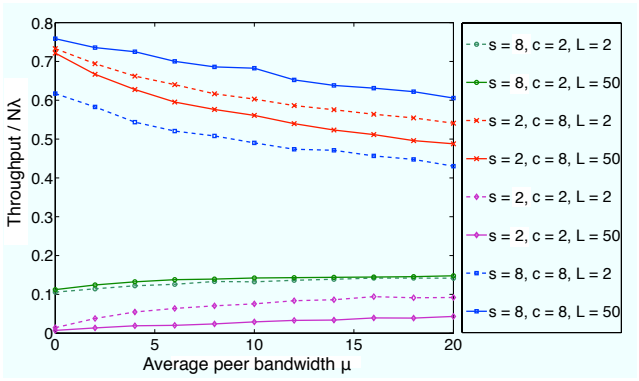

Figure 4. Session throughput as a function of μ under different scenarios. $\lambda = 8$, $\gamma = 1$.

We also simulate to study the session throughput in a dynamic network. Peer dynamics is simulated via a replacement model [7], [8], where each peer is assigned a random lifetime $\boldsymbol{L}$ and leaves the network upon the expiration of its lifetime. A new peer will join at the same time to replace the departed peer. The peer lifetime follows an exponential distribution with mean L. Such a model allows us to decouple the impact of the change in the number of online peers and fully focus on the effect of their *dynamic* nature.

The impact of churn on session throughput is plotted in Fig. 4 under various parameter settings. When server capacity compares to user demands ($c = 8$), throughput is degraded for the case of severe peer churn (dashed lines) as a larger segment size is used, and as more buffering is performed with a larger μ. This is because, buffering by gossiping and network coding is actually not needed when server capacity is high enough, whereas the use of a larger segment size in this case could make segments become undecodable when peers abort too frequently. However, when server capacity is insufficient, i.e., c/λ is small, servers cannot collect all the generated data immediately anyway. Introducing a higher data redundancy will aid the collection of these data in a delayed fashion before they disappear. Thus, when server capacity is insufficient, throughput benefits from having a larger segment size s and higher peer bandwidth μ, even in the presence of peer churn, as shown in Fig. 4.

We define *block delay* as the delivery delay of a segment

divided by the segment size, that is the average delivery delay of each original block. We can derive block delay using Little's Theorem in queueing theory based on the session throughput.

Theorem 3 *(Block Delivery Delay) In steady-state network, if an original block can eventually be reconstructed at the server, the average time from the injection of this block to its delivery is*

$$T(s) = \frac{\sum_{i=1}^{\infty} \tilde{w}_i}{\lambda} - \frac{\sum_{i=1}^{\infty} \tilde{m}_i^s}{\lambda \sigma(s)}, \tag{17}$$

where $\tilde{w}_i$ is the steady-state solution to (8), $\sigma(s) = Throughput(s)/N\lambda$.

Proof: Let $\sigma(s) = Throughput(s)/N\lambda$. Let T_L denote the average time from the injection of a segment to its extinction from the network. By Little's Theorem, $T_L = \sum_{i=1}^{\infty} \tilde{X}_i / \frac{N\lambda}{s} = s\sum_{i=1}^{\infty} \tilde{w}_i/\lambda$, where $\sum_{i=1}^{\infty} \tilde{X}_i$ is the number of distinct segments in the network and $N\lambda/s$ is segment injection rate. We call a segment a *good segment* if the segment has been pulled by the servers for s times and is still available in the network. Similarly, the average time during which a segment is good is $T_M = \sum_{i=1}^{\infty} \tilde{M}_i^s / \frac{N\lambda\sigma}{s} = s\sum_{i=1}^{\infty} \tilde{m}_i^s/\lambda\sigma$. Hence, the average block delay $T = (T_L - T_M)/s = \sum_{i=1}^{\infty} \tilde{w}_i/\lambda - \sum_{i=1}^{\infty} \tilde{m}_i^s/\lambda\sigma(s)$. □

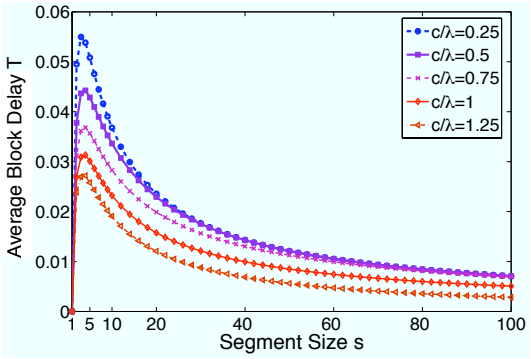

Figure 5. Average block delivery delay T for different values of s. $\lambda = 20$, $\mu = 10$, $\gamma = 1$.

Fig. 5 shows the numerical results on block delay. We can see that there is a peak of block delay around $s = 5$. The reason is that without coding ($s = 1$), although many blocks are lost due to a low throughput, yet a block is collected immediately if it can be collected as servers do not have to buffer a sufficient number of (coded) blocks to reconstruct a segment. When a larger segment size is used, servers tend to collect blocks from different segments in an alternating fashion, resulting in a longer delay to reconstruct either one of them. However, when s is sufficiently large, the delay decreases again, as blocks in the network tend to belong to a few large segments. Taking into consideration of both throughput and delay, a segment size between 20 and 40 is preferred.

Finally, we show that at the end of the stream collection session, there is a guarantee of a certain amount of data buffered in the network so that the servers can still collect them in a delayed fashion.

Theorem 4 *Suppose the streams of upload requests end in steady-state network, then the amount of data in terms of original blocks saved up in the network for future delivery is*

$$S = N \cdot s \sum_{i=s}^{\infty} (\tilde{w}_i - \tilde{m}_i^s), \tag{18}$$

where $\tilde{w}_i$ is the steady-state solution to (8).

Proof: The number of segments buffered for future delivery equals to the number of decodable segments that have not been reconstructed by the servers yet. The number of decodable segments is $\sum_{i=s}^{\infty} \tilde{X}_i = N \cdot \sum_{i=s}^{\infty} \tilde{w}_i$. Among these segments there are $\sum_{i=s}^{\infty} \tilde{M}_i^s = N\sum_{i=s}^{\infty} \tilde{m}_i^s$ reconstructed by the servers already. Upon the decoding of a segment, s original data blocks are reconstructed at the servers. Thus, we have $S = N \cdot s \sum_{i=s}^{\infty} (\tilde{w}_i - \tilde{m}_i^s)$. □

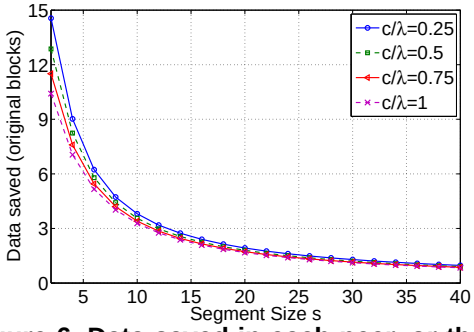

Figure 6. Data saved in each peer, or the average number of original blocks buffered in each peer that have not been reconstructed by the servers. $\lambda = 20$, $\mu = 10$, $\gamma = 1$.

We plot the number of original blocks buffered in each peer that have not been reconstructed by the servers in Fig. 6. This amount of data is saved up for future delivery when statistics data are streamed at each peer at a much lower rate. According to Theorem 1, the total amount of data buffered in the peers are the same regardless of the segment size used. However, as the segment size increases, network throughput increases according to Theorem 2. As more data in the network are already reconstructed in the stream collection session, each peer buffers less "fresh" segments that have not been reconstructed by the servers. However, regardless of the segment size, the system benefits by using an indirect collection mechanism which always buffers a *guaranteed* amount of data for future delivery when the volume of traffic falls down.

5 Related Work

The problem of statistics collection in distributed systems has been approached with various methods before. Stutzbach *et al.* [10] design a crawler to capture snapshots of Gnutella network, which focuses on increasing snapshot accuracy by increasing crawling speed. Such an approach leverages the two-tier topology of Gnutella networks and is difficult to be generalized to arbitrary network topologies. NetProfiler [9] is a peer-to-peer infrastructure proposed for profiling wide-area networks, which aggregates information along DHT-based attribute hierarchies and thus may not adapt well to high peer churn rates. Astrolabe [11] aggregates information for distributed system monitoring, with gossip-based information distribution and replication. Echelon [13] leverages the power of network coding to collect application-specific measurements on each peer, and disseminate them to other peers in a coded form.

Our work differs from all previous work in two aspects. First, our algorithms aim at providing *Quality of Service (QoS) guarantees* in face of peer dynamics and fluctuating traffic. Second, we demonstrate such QoS guarantee using both theoretical analysis and simulations, while most previous work is only experimental. Such a modeling effort with theoretical rigor helps us to draw more insights than mere experimental work can do.

Network coding has been introduced for distributed storage in [3, 4] for a sensor network scenario. The idea has subsequently been extended to P2P storage [2]. In our paper, we utilize a similar idea to spread coded blocks across the network from which servers pull data. However, the focus of this paper is not to maintain a long-lasting distributed storage system. Instead, to keep a low storage overhead, the buffered data at each peer are only ephemeral with short lifetimes. Our goal is to circumvent the server bottleneck suffered by the centralized logging server approach, using a best-effort indirect collection scheme that probabilistically collects as much data as possible from the network.

6 Concluding Remarks

In most large-scale peer-to-peer (P2P) applications, it is necessary to collect statistics data to diagnose system performance and network outage. With current logging server solutions, such statistics collection is not scalable to up to millions of participating peers, due to the bottleneck at servers. In this paper, we propose an indirect statistics collection scheme based on network coding to collect large volumes of data in a delayed fashion. We analyze the system performance via a random bipartite graph process and show that by utilizing peer resources to buffer data in a decentralized way, the server bandwidth needs to be provisioned to handle only *average* load rather than the *peak*. Furthermore, as data blocks are distributed across the network, the servers can always retrieve buffered data in a delayed fashion that they were not able to collect during a flash crowd scenario.

References

[1] P. A. Chou, Y. Wu, and K. Jain. Practical Network Coding. In *Proc. of 41th Annual Allerton Conference on Communication, Control and Computing*, October 2003.

[2] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran. Network Coding for Distributed Storage Systems. In *Proc. of IEEE INFOCOM*, 2007.

[3] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Ubiquitous Access to Distributed Data in Large-Scale Sensor Networks through Decentralized Erasure Codes. In *Proc. of IPSN*, 2005.

[4] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran. Decentralized Erasure Codes for Distributed Networked Storage. *IEEE Transactions on Information Theory*, 52(6):2809–2816, June 2006.

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

[6] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros. The Benefits of Coding over Routing in a Randomized Setting. In *Proc. of ISIT*, 2003.

[7] D. Leonard, V. Rai, and D. Loguinov. On Lifetime-Based Node Failure and Stochastic Resilience of Decentralized Peer-to-Peer Networks. In *Proc. of ACM SIGMETRICS'05*, Banff, Alberta, Canada, June 2005.

[8] D. Niu and B. Li. On the Resilience-Complexity Tradeoff of Network Coding in Dynamic P2P Networks. In *Proc. of IWQoS 2007*, Evanston, Illinois, USA, June 2007.

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

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

[11] R. van Renesse, K. Birman, and W. Vogels. Astrolabe: A Robust and Scalable Technology for Distributed System Monitoring, Management and Data Mining. *ACM Transactions on Computer Systems*, 21(2):164–206, May 2003.

[12] N. C. Wormald. Differential Equations for Random Processes and Random Graphs. *The Annals of Applied Probability*, 5(4):1217–1235, November 1995.

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

[14] C. Wu, B. Li, and S. Zhao. Magellan: Charting Large-Scale Peer-to-Peer Live Streaming Topologies. In *Proc. of of ICDCS 2007*, Toronto, Ontario, Canada, June 2007.

[15] C. Wu, B. Li, and S. Zhao. Multi-channel Live P2P Streaming: Refocusing on Servers. In *Proc. of IEEE INFOCOM 2008*, Phoenix, Arizona, April 2008.