Scalable and Efficient Data Streaming Algorithms for Detecting Common Content in Internet Traffic

Minho Sung, Abhishek Kumar, Li (Erran) Li, Jia Wang, and Jun (Jim) Xu

Abstract— **Recent research on data streaming algorithms has provided powerful tools to efficiently monitor various characteristics of traffic passing through a single network link or node. However, it is often desirable to perform data streaming analysis on the traffic aggregated over hundreds or even thousands of links/nodes, which will provide network operators with a holistic view of the network operation. Shipping raw traffic data to a centralized location (i.e., "raw aggregation") for streaming analysis is clearly not a feasible approach for a large network. In this paper, we propose a set of novel Distributed Collaborative Streaming (DCS) algorithms that allow scalable and efficient monitoring of aggregated traffic without the need for raw aggregation. Our algorithms target the specific network monitoring problem of finding common content in the Internet traffic traversing several nodes/links, which has applications in network-wide intrusion detection, early warning for fast propagating worms, and detection of hot objects and spam traffic. We evaluate our algorithms through extensive simulations and experiments on traffic traces collected from a tier-1 ISP. The experimental results demonstrate that our algorithms can effectively detect common content in the traffic traversing across a large network.**

I. INTRODUCTION

In recent years, the problem of monitoring and analyzing the aggregate traffic passing through many high-speed links has emerged as an important and challenging problem in network measurement and management. Monitoring the characteristics of this aggregate traffic is essential for detecting "global" events that are intrinsically distributed through the network. Examples of such events range from global top-k traffic sources (global elephants) to incipient worm infections (involuntarily "popular" content). It is hard to detect such events using traditional per-link monitoring mechanisms since the signal is usually too feeble to be detected locally. Such events may leave indelible signatures in the aggregate traffic, but only through correlation of traffic among many links can this signature be revealed.

In this paper, we focus on a specific problem in monitoring aggregated traffic – detecting common content in the packet-level traffic across multiple network links. Note that although a content may be widely spreading across a network, local monitoring at a single point in the network might fail to identify such content since the frequency with which packets containing the same application layer data pass through any unique monitoring point (link or node) in a network might not be significant.

We propose a set of novel distributed data streaming algorithms that allow us to perform large-scale distributed measurement on tens of thousands of high-speed links and nodes. Data

M. Sung, A. Kumar, and J. Xu are with College of Computing, Georgia Institute of Technology, L. Li is with Bell Labs, Lucent Technologies, and J. Wang is with AT&T Labs – Research. E-mails: {mhsung,akumar,jx}@cc.gatech.edu, erranlli@bell-labs.com, jiawang@research.att.com

streaming is concerned with processing a long stream of data items (e.g., packets) in one pass using a small working memory in order to answer a type of query regarding the stream. The trick is to remember, in this small memory, information that is most pertinent for answering the query. Our solution, extends this data streaming vision to distributed monitoring as follows. Each link first processes traffic going through it using streaming algorithms specialized for gathering fragments that may potentially become a part of the signature we are looking for in the aggregate traffic. These streaming results, which are several orders of magnitude smaller than the original traffic stream, will be shipped to a data processing center for synthesis and analysis to detect common content. The data processing center needs to correlate different fragments and identify the common content signature superimposed with "background noise". We will show that this task is very challenging since the signals are so weak that novel signal processing techniques have to be developed to magnify and detect it. We demonstrate through extensive simulations and stress tests using traffic traces collected from a tier-1 ISP that our algorithms are able to detect common content "planted" in the Internet traffic very effectively. To our best knowledge, this is the first set of distributed data streaming algorithms for network monitoring and measurement.

A. Motivations for detecting common contents

Rapid spreading of common content is a daily phenomenon in today's Internet. A number of traffic flows carrying the same application layer data can often be seen along different paths across a network. Examples of such content include popular Web pages, "hot" music files, Internet worm/virus files, and spam emails.

Web browsing. Even with the deployment of Web proxies and content distribution networks (CDNs), a significant number of duplicated content (not necessarily from the same URL) are delivered over the Internet, especially in a flash crowd event. Detecting common content being transmitted in Web traffic flows might help network operators to react to such flash crowd events.

P2P file sharing. P2P file-sharing has become one of the most popular applications. Content is shared (illegally in most cases) among all the users. Hot content, e.g., newly released movies, is likely to be requested by many users. As a result, such content can be transmitted to many destinations over the Internet, consuming a large amount of bandwidth. Monitoring common content delivered to different users can allow us to track illegal content sharing.

Internet worm/virus. Internet worms (non-polymorphic) and viruses also have the flavor of widely spreading common con-

tent[1]. Identifying common content across multiple links may help us identify an *unknown* worm that is in its earlier stage of propagation. This may potentially win us a couple of critical hours to effectively control the damage.

Spam emails. Unsolicited email or spam is a significant consumer of network resources. In this case, copies of the same message are sent to many users simultaneously. Except for the SMTP header, the body of the messages would be the same in all the instances. Detecting spam emails would help operators to set up proper filters to block the unwanted spams.

We concede that illegal or malicious content can easily evade detection through various obfuscation techniques. For example, (illegal) P2P content can be encrypted with different keys to look different in each distance; worms and viruses can change their content or use encryption; spam emails can have random gaps between words to fail any string matching effort. We argue, however, that even in this context our effort serves a purpose for the following reason. First, obfuscation often affects the effectiveness of the malicious content in terms of propagation and infection. For example, key distribution required for encrypting content with different keys in order to evade our detection clearly increases the complexities of such activities; encrypted worms and viruses may not work well on systems that are not equipped with the decryption algorithm and correctly implementing polymorphic worms might require higher levels of programming skills. Finally, we expect that, detecting common content as a well-defined Internet monitoring primitive, may find new applications in the future Internet.

B. Paper outline

The rest of this paper is organized as follows. In Section II, we describe an overview of our solution framework and streaming algorithms. We present the algorithm for detecting common content in a distributed manner in Section III and Section IV. Evaluation of algorithms using simulations and experiments on traffic traces collected from a tier-1 ISP is presented in Section V. Finally, we present related work in Section VI and conclude in Section VII with directions to future work.

II. OVERVIEW

In this section, we give the problem statement and describe an overview of our solution framework. We then provide an intuitive description of our detection algorithms and discuss some of the subtleties involved in detecting common content. We conclude this section with a discussion of the assumptions used in the rest of the paper.

A. Problem statement

We view an object/file as a string, and say that two objects/files share the same content if they contain a large common substring. We consider two cases in Figure 1.

Aligned case. Two instances of objects/files are simply identical. Same content encountered in Web browsing, P2P file sharing, and some Internet worms such as CodeRed and Slammer are examples of this category.
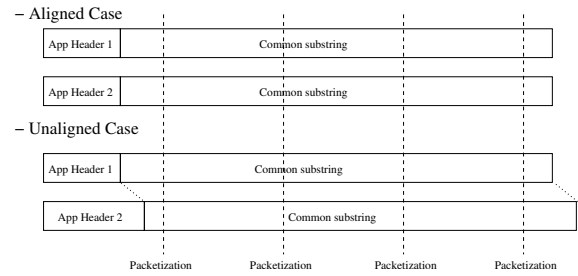
[1]Our solution cannot detect a polymorphic worm which changes its binary content during infection/propagation.

Fig. 1
THE ALIGNED AND UNALIGNED CASES.

Unaligned case. The common substring starts at different locations in the object. In other words, there is a variable prefix that precedes the common substring. Many email worm viruses such as Nimda, Sircam, or Mimail are examples of the unaligned case. Due to the nature of SMTP, the size of the application-layer header before the fixed attachment of the content are variable. For example, the application layer portion of a worm could contain some information depending on the specific identities of victims. In a worm propagating as an email attachment, the application layer interpretation of the worm string is that of an email with one or more attachments. The initial portion of this string would be the SMTP header for the email with fields like "From", "To", and "Reply-to" that would vary from one instance of the worm to the other.

In the observed traffic flows, an object is packetized into one or more IP packets. If these packets are of a fixed size except for the last packet and this size is the same for two identical objects A and B, the i_{th} packet of A will have the same payload as the i_{th} packet of B in the aligned case. Under the same assumption, packets from two identical objects in the unaligned case will result in the same packet content under a "shift" according to the difference in the initial offset.

B. Solution framework

The problem of detecting content that is common across a large set of nodes is essentially equivalent to detecting common substrings in the aggregate traffic traversing all of these nodes. However, traditional string-matching algorithms are too slow to operate on the immense scale of data flowing through today's large networks. In fact, any centralized solution, that requires all the raw data to be aggregated at one location for analysis, would be impractical due to the unaffordable logistics of shipping all the data to a data processing center. For example, aggregating 1000 OC-192 ingress links results in an aggregated traffic stream of 10Tbps, which would require doubling the network capacity just to ship a copy of all the traffic to the central analysis station. Clearly, any practical solution needs to be lightweight and distributed. Next, we describe a solution framework for our solution that fits this bill.

The overall architecture of our solution is shown in Figure 2. The idea is to use lightweight data-collection modules running at each monitored link or node to process the cross traffic at line speeds and produce small digests that can be shipped to a central analysis module for further processing. The data collection modules uses specially designed data-streaming algorithms to produce succinct digests that are 1000 times smaller than the
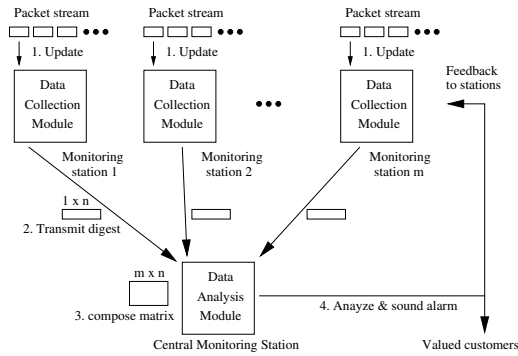
Fig. 2
SOLUTION FRAMEWORK.

processed traffic, making it affordable to ship the digests to a central analysis module. A synthesis algorithm at the analysis module aggregates these digests and processes them together to detect common content.

The challenge in this solution framework is to design the local streaming algorithms and the data synthesis algorithm in such a way that the digests they produce contain information, *pertinent to the events we are looking for*, with an accuracy almost as good as obtainable from processing the aggregated traffic directly. The algorithms to be presented in the following sections employ sophisticated techniques to extract as much information as possible from these digests. This approach places stringent requirements on the design of both modules:

Data collection module. First, the collected data has to be much smaller than the original raw data size. Our algorithm is expected to achieve at least three orders of magnitude reduction on the traffic volume. Second, the collected data has to contain enough information for accurate analysis of the traffic. These two are conflicting requirements that are finely balanced in our design. Third, the data collection mechanism should be fast enough to keep up with high line speeds of 10 Gbps (OC-192) or even 40 Gbps (OC-768).

Data analysis module. For continuous monitoring, the data synthesis algorithm has to be able to process each second's worth of traffic in one second. However, since these algorithms are easily parallelizable, this requirement can be relaxed when we have tens of CPU's to use. Within this computational complexity constraint, our algorithms need to identify patterns from these digests, with both low false positive (report a pattern that does not actually exist) and low false negative (fail to report a pattern).

C. Algorithm overview

We provide an intuitive description of our mechanisms here, deferring a detailed treatment to the next two sections. The data collection modules collect specially constructed bitmaps, succinctly preserving signatures of the strings seen in the actual traffic. The analysis algorithm then tries to discover *correlations* among the various bitmaps collected at the distributed monitoring points. We need to find the bitmaps that share exactly the same content signature as well as the signature in these bitmaps. However, in our formal treatment in section III-B, the most general form of this problem, finding the largest submatrix

of all ones in a 0-1 matrix, turns out to be NP-hard. The running time of the best polynomial-time approximation algorithm for the general problem in the literature [1] can not meet our design goal. Fortunately, in our special setting, the input of the problem is not arbitrary, it is the superposition of common content signature and values of random variables that are approximately Bernoulli. Exploiting this property of the bitmap-construction procedure, we design a much more efficient polynomial time algorithm.

The unaligned case, where different initial offsets can cause the same piece of content to be packetized differently in individual instances, turns out to be slightly more complex providing us the grounds to design more sophisticated detection schemes. Due to the variable offsets, bitmap construction at data collection modules needs more complexity to capture signatures of randomly shifted content. To borrow an analogy from signal processing, we need to *amplify* the signal because it is weakened due to the presence of *noise* (random offsets). We introduce techniques that perform such amplification during data collection in Section IV.B.

Detecting correlations among such complex bitmaps is also less straightforward. In section IV-B, we design a novel technique that uses phase changes in Erdös-Renyi random graphs to detect such correlations. The phase-change theorem for Erdös-Renyi random graphs says that if the probability of the existence of an edge between any two of the n vertices in such a graph is less than $\frac{1}{n}$, then, with high probability, all connected components are of size $O(\log n)$. However, when this probability is greater than $\frac{1}{n}$, a giant connected component of size $\Theta(n)$ begins to emerge. The design of our detection algorithm leverages this theorem in the following manner. First, the pairwise correlation among various bitmaps is computed. We then construct a graph with vertices representing bit-vectors and impose edges between a pair of vertices according to p – an appropriately scaled value of the correlation between the corresponding bitmaps. The scaling factor is chosen in such a way that the expected value after scaling is below $\frac{1}{n}$ if there is no common content. This would result in a graph with small connected components. However, if common content is present, with the same scaling coefficient, the value of p increases beyond $\frac{1}{n}$ for pairs of nodes where both nodes have seen the common content. This increase in p is due to the high correlation between bitmaps collected at two nodes that have both recorded the passage of the same piece of content. The global effect of this increased value of p is that the size of the largest connected components in the Erdös-Renyi random graph becomes much larger than $O(\log n)$, indicating the the presence of common content. As we show later, this simple test turns out to be miraculously accurate.

Once this Erdös-Renyi test (described in detail in Section IV-B) indicates the presence of a pattern, we need to identify the actual nodes that saw the common content. Formally, the general problem is equivalent to finding a maximum clique. This problem is NP-hard and there does not exist any constant factor approximation algorithm for it [2]. Our graph is mostly the union of a random graph and some "clique-like" dense subgraphs. Using this property, we propose a greedy algorithm to find most of the vertices in the largest *cluster*, i.e., the largest

set of vertices that connect to each other with higher probability than $\frac{1}{n}$. The algorithm is proven to be stochastically optimal under a reasonable computational constraint, using stochastic ordering theory.

D. Assumptions

In the rest of the paper, we assume that common content is always chopped up into packets of the same size. Our algorithms can be extended to cover the more general case of variable packet-sizes, but we make this assumption for simplicity of presentation. This assumption is justified by the following reason. Typically the same application-layer protocol is used to transmit such common content, e.g., email viruses are always transmitted over SMTP. If the application runs over TCP, it typically adopts the standard MSS (Maximum Segment Size). A recent study of Internet packet size distribution [3] indicates that there are only 2 popular packet sizes no smaller than 500 bytes: 576 and 1,500 bytes. So in practice, a large portion of common content will be transmitted over the same packet size. In this paper, we focus on common content transmitted using such popular size. We remark that, our algorithms work for common content transmitted with any packet sizes; however, they are optimized for the common case.

In addition, our algorithm can be viewed as a clustering algorithm which detects one large cluster in the dataset. This cluster can contain either single common item or multiple common items. The techniques that are used to separate out sub-clusters upon detecting a large cluster have been maturely developed. Thus we will focus on only detecting one large cluster assuming those techniques can be used on top of our algorithm to report multiple common content occurring within the same measurement epoch.

III. DESIGN FOR THE ALIGNED CASE

In this section, we describe our distributed streaming algorithm for detecting common content in Internet traffic for the aligned case, deferring the unaligned case to the Section IV.

A. Distributed online streaming module

Our algorithm for the online streaming module, which balances two conflicting demands of operational efficiency and detection accuracy, is illustrated in Figure 3. The data structure used in the data collection phase of our architecture is extremely simple – a hashed bitmap, i.e., an array of n bits that is indexed by a uniform hash function with range equal to the size of the array. The bitmap is set to all 0's at the beginning of a measurement epoch. When a packet arrives, we strip the network and transport layer headers to obtain the application layer data, represented as a string. The hash function associated with the bitmap is applied on this string or a part of it[2] to produce an index into the bitmap, and the corresponding bit is set to 1.

The size n of the array is determined such that it holds one measurement epoch's traffic for a router running at reference

---

[2]Note that $range(pkt.content, 0, len)$ in line 5 of Figure 3 refers to the first len bytes of the packet content. We hash only packets which actually contain payloads.

```
1. Initialization
2.     A[i] := 0, i = 1, · · · , n

3. Update algorithm
4.     Upon the arrival of a packet pkt
5.         index := hash(range(pkt.content, 0, len));
6.         A[index] := 1;
```

Fig. 3

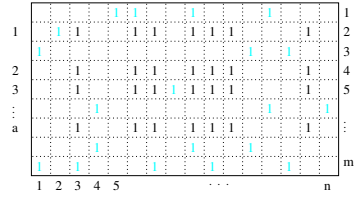BASIC ALGORITHM FOR UPDATING THE ONLINE STREAMING MODULE.

Fig. 4

COMPOSED $m \times n$ BITMAP AFTER RECEIVING $1 \times n$ DIGEST FROM m ROUTERS: THE BITMAP HAS AN $a \times b$ PATTERN. (BOLD AND GRAY 1'S REPRESENT THE PATTERN AND NOISES RESPECTIVELY)

speed. Throughout this paper, we are targeting the link speed of OC-48 (i.e., 2.4 Gbps). We assume a conservative average packet size of 1,000 bits. Larger packet sizes will clearly serve to our advantage since there will be fewer packets in a measurement epoch. Running at full capacity, each router line-card will process at most 2.4 million packets in each one second epoch. Since putting $(\ln 2)l$ random bits into an l bit array will make the array contains approximately half 1's and half 0's (the property of Bloom filter [4]), an array of about 4 million bits can fit about 2,907,269 packets, enough for a little more than 1 second's traffic on an OC-48 link. For this reason we set the width of the bitmap to 4 megabits.

B. Data analysis module

The data collection algorithm running at each measurement point converts the packet stream in a measurement epoch into an n-bit bitmap (i.e., $1 \times n$ matrix). Once about half of the n bits become 1's, the measurement epoch ends and the bitmap is sent to the centralized monitoring station for analysis. Bitmaps from m measurement points will form an $m \times n$ matrix. The rows of the matrix correspond to different routers and the columns correspond to different packets. In this matrix, a 1 in the i-th row j-th column corresponds to the i-th router seeing a packet that hashed to the index j. Common content consisting of b packets that is seen by a routers corresponds to a non-contiguous submatrix of size $a \times b$ in which all entries are 1's (Figure 4). However, the reverse is not true as the bitmaps could have false positive[3] (i.e., bits are set to 1's due to collisions in hashing). Now our pattern detection problem can be formulated as follows. Given an $m \times n$ 0-1 matrix, we would like to find an $a \times b$ submatrix that are all 1's, referred to as All-1 Submatrix IDentification (ASID) problem. The general case of ASID problem, unfortunately, is NP-hard.

Theorem 1: The ASID problem is NP-hard.

Proof: The proof is via reduction from the Maximum Edge Biclique problem. The Maximal Edge Biclique problem (MBP) is stated as follows. Given a bipartite graph $G =$

---

[3]This effect is accurately captured in our later analysis.

$(V_1 \cup V_2, E)$ and a positive integer K, does G contain a bi-clique with at least K edges? The MBP problem is known to be NP-hard [5]. For each node u in V_1, we create a router. For each node v in V_2, we create a packet. We index the packets as $1, 2, \cdots, |V_2|$. If node $v \in V_1$ sees packet i, we set the bit array position i to 1. It is easy to see that there is a biclique of size K in G iff there is a submatrix with K entries and all entries are 1. ∎

Fortunately, our problem of detecting the common content is more tractable than the general case because the $m \times n$ matrix cannot take arbitrary values: each element in the $m \times n$ matrix in our scenario is drawn from a nice probability distribution (Bernoulli in our scenario). Therefore, we are able to find a polynomial greedy algorithm that identifies such submatrix with high probability.

In the following, we first state a naive algorithm to motivate our solution. The complexity of the naive algorithm is $O(n^2 \log n)$, where n is the number of columns in a matrix. This complexity is still way too high since n is typically in the millions when the link speed is as high as OC-192 and OC-768. We then propose a refined algorithm of complexity $O(n \log n)$ that has almost the same prediction accuracy as the naive algorithm.

We refer to the bit-wise "AND" of two column vectors, the result of which is a vector, as their *2-product*. Extending this definition, we refer to the result of bit-wise "AND" of k columns as their k-product. We omit this 2 or k before the "product" whenever it is clear from the context. For a given k-product v, we denote the set of k column vectors that resulted in v as A_v. We refer to the number of 1's in a column vector v as its weight (denoted as $weight(v)$).

The naive algorithm. Our naive algorithm is illustrated in Figure 5. Recall that the problem of finding $a \times b$ all-1 submatrix is equivalent to find a set of at least b columns such that the weight of their b-product is no smaller than a. A brute-force algorithm to find these b columns is prohibitively expensive since it would have to exhaustively try all $\binom{n}{b}$ combinations. Our naive algorithm, with complexity only $O(n^2)$, is to use a greedy algorithm to find such $a \times b$ submatrices *with high probability* as follows. We first computes the 2-products for all pairs of columns, a total of $\binom{n}{2} = \frac{1}{2}n(n-1)$ of them. The complexity of this step is clearly $O(n^2)$ bitwise-AND operations on vectors. However, instead of storing all these results, the algorithm only maintains the heaviest $O(n)$ 2-products as "hopefuls" using a priority queue of this size. The cost of maintaining this queue with $O(n^2)$ insertions is $O(n^2 \log n)$. These "hopefuls" will bit-wise "AND" with all n columns in S_1 to result in no more than $O(n^2)$ 3-products, among which the heaviest $O(n)$ entries are maintained again using a priority queue. This step[4] again has complexity $O(n^2 \log n)$. Then the $O(n)$ heaviest 3-products will become the "hopefuls" to generate the 4-products and so on. This iteration process is shown in line 2 through 4 in Figure 5. We will show next that the number of iterations of this "hopeful" selection process can be viewed as a constant. Therefore, the total complexity of the naive algorithm is $O(n^2 \log n)$.

---

[4]Note that in this step we omit all 3-products in which the same column has appeared more than once.

```
Initialization
  1.   Let S₁ be the set of column vectors (|S₁| = n)

Finding the pattern
  2.   for ( b′ = 2 ; b′ ≤ num_iterations; b′++ )
  3.      /* num_iterations explained later in text */
  4.         S_b′ = O(n) heaviest vectors among all b′-product
                  v · w where v ∈ S_{b′−1}, w ∈ S₁ and w ∉ A_v
  5.      v = heaviest vector in S_{number_iterations}
  6.      If (weight(v) × b′ is non-naturally-occurring pattern) then
  7.         output A_v
  8.      Else declare no pattern exist
```

Fig. 5
NAIVE DETECTION ALGORITHM FOR THE ALIGNED CASE

In line 2 of Figure 5, we set the number of iterations to a constant $num\_iterations$ for the simplicity of discussion. In reality, however, we do not know this number a priori. The actual number of iterations is determined through a procedure that checks a termination condition, which we will describe after we present the refined algorithm. In fact, the actual logic for termination is as follows. We run the loop up to $num\_iterations$ (an upper bound) times. If the termination condition is met, the loop is terminated prematurely and reports that a pattern is found. Otherwise, after $num\_iterations$ iterations, the loop terminates and reports that there is no pattern. Next, we will show that this upper bound $num\_iterations$ needs to be no more than $b + c$, where c is a small constant. It can be shown that for the $a \times b$ pattern to be non-naturally occuring [5], b should be about $O(\log m)$, where m is the number of rows (routers). When m is upper bounded by a constant (say 10,000), we can view b as a constant. Therefore, we can view the number of iterations we need as upper-bounded by a constant.

Our Monte-Carlo simulations show that picking $O(n)$ in every iteration is *sufficient* for the pattern to "survive" and be detected by the above algorithm with high probability. We do not know, however, whether $O(n)$ is *necessary*. We stick to $O(n)$ because the first iteration already has complexity $O(n^2 \log n)$. Even if we make the "hopeful" list shorter, the asymptotic complexity is not going to drop significantly. Note that, we have used the O notation for certain parameters in our algorithm. The constant will be determined by parameter tuning for the specific setting in practice. Note this tuning is done *once and for all* given the number of routers m and the size of the bitmap n.

Finally, after these iterations, the heaviest product will contain a subset of the pattern with high probability. If this pattern is non-naturally occuring (line 6), we declare that we have found a part of the pattern and output it (line 7). Otherwise, we declare that no pattern is found (line 8).

The refined algorithm. The refined algorithm shown in Figure 6 improves the performance of the naive algorithm. The tradeoff is that patterns have to be much larger to be detectable. Our idea is that, instead of searching for the pattern in the original matrix, we search for a much smaller subpattern (lines 2 to 5), referred to as a "core", in a submatrix consisting of a subset of the columns, referred to as S_1. This core, if found, will be used to scan through all the columns not in this submatrix to help identify the whole pattern (lines 10 through 14). We

---

[5] A pattern is *non-naturally occurring* if the probability for it to exist in the data that does not contain any common content is below a threshold ϵ. More detail will be given in Section III-C.

```
Initialization
  1.  Let S be the set of column vectors, |S| = n
  2.  Let S₁ be the n′ heaviest column vectors in S

Finding the small subset of the pattern
  3.  for ( b′ = 2; b′ ≤ num_iterations; b′++ )
  4.    /* num_iterations explained later in text */
  5.    S_{b′} = n′ / log n′ heaviest vectors among all b′-product v · w
            where v ∈ S_{b′−1}, w ∈ S₁ and w ∉ A_v
  6.    Core = heaviest vector in S_{number_iterations}
  7.    If (weight(Core) ×b′ is non-naturally-occurring pattern) then
  8.      go to greedy addition routine
  9.  declare no pattern exist

Greedy addition of vectors into the pattern
 10.  thresh = weight(Core) − γ
 11.  S* = A_{Core}
 12.  ∀w ∈ S, if weight(w · Core) ≥ thresh
 13.    S* = S* ∪ w
 14.  Output S*
```
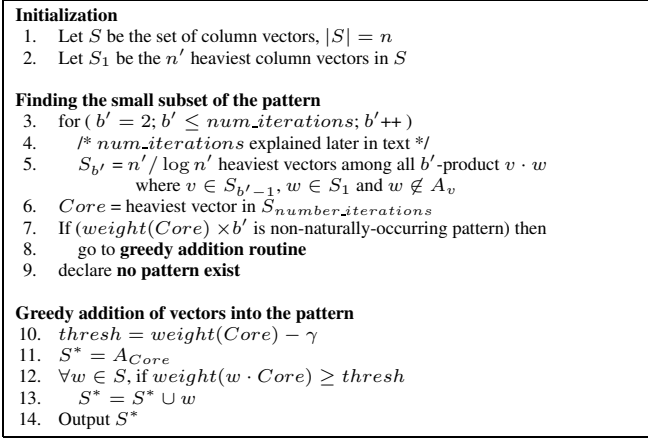
Fig. 6

REFINED DETECTION ALGORITHM FOR THE ALIGNED CASE

choose the heaviest n' columns to be the submatrix because a heavier column is more likely to be a part of the pattern. The value of n' is determined using the theorem 2 in the following, the proof of which is omitted for lack of space. The main idea of this theorem is to ensure that n', the size of S_1, is large enough such that, with high probability, S_1 will contain a subpattern large enough ($a \times l$) to be non-naturally-occuring in the submatrix consisting of columns in S_1. In our context, when n is in the range of millions, we found that n' only needs to be in the range of thousands. Therefore, for the range of values of the parameter n in our mechanism, n' is $O(\sqrt{n})$, and the overall complexity of the refined algorithm is reduced to $O(n \log n)$. We have not verified, however, whether n' is indeed $O(\sqrt{n})$ for all n. This can be an interesting topic for future research.

Theorem 2: Suppose there is only one pattern of size at least $a \times b$. Let the number of rows be m and the number of columns be n. Let w and s be thresholds such that $binocdf(w, m, 0.5) = 1 - \epsilon_1$ and $binocdf(s, n, \epsilon_1) = 1 - \epsilon_2$. Here $binocdf(x, n, p)$ denotes the probability $Pr[X \leq x]$ where the random variable X follows the binomial distribution with parameter n (the number of trials) and p (the probability of success for each trial). Suppose $binocdf(w - a, m - a, 0.5) = 1 - \epsilon_3$ and $binocdf(l, b, \epsilon_3) = 1 - \epsilon_4$. Let S_1 be a set of $s + b$ heaviest columns. In other words, $n' = s + b$. Then, the probability that S_1 contains at least l columns belonging to the pattern is at least $1 - \epsilon_4 - \epsilon_2$.

After we get the core, we greedily add more columns into the witness set S^* as illustrated in lines 10 to 14 of Figure 6. Since the core can contain some rows (routers) that are erroneously identified as belonging to the pattern due to the noise, we define a threshold $thresh = weight(Core) - \gamma$ where γ is a tuning parameter. If a vector $v \in S - S_1$ has more than $thresh$ number of common ones with $Core$, v will be considered a part of the pattern. With our typical parameters, setting γ to 2 or 3 will work very well almost 100% of time.

Termination procedure. Recall that we set the number of iterations to a constant $num\_iterations$ for the simplicity of presentation. We now describe the aforementioned termination procedure that is used to determine when to stop the iterations (prematurely) in the naive algorithm (also used in the refined algorithm). Its basic idea is, for each iteration of b', the weight
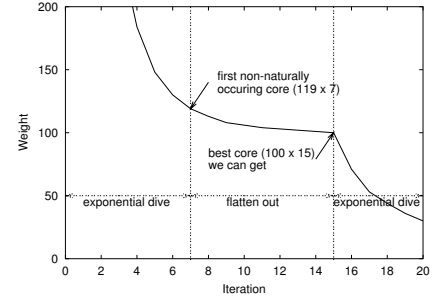

Fig. 7

$weight(Core)$ AND $|S^*|$ AT EACH STAGE

of the heaviest b'-product will go down almost exponentially at the very beginning because the rows not contained in the pattern (noise) will be zeroed out in the b'-product by slightly less than 50% in each iteration. If the matrix contains no pattern, this exponential decay will continue till the weight becomes zero. On the other hand, if the matrix indeed contains a pattern, this "weight loss" will flatten out gradually because the noise by now is mostly gone and the currently heaviest b'-product absorbs more and more columns that belongs to the pattern. Finally at a certain point the "weight loss" becomes exponential again. This happens when all columns that belong to the pattern have been absorbed and each absorption of a column that does not belong to the pattern will again result in slightly less than 50% "weight loss" in each iteration. *Our program should terminate right before the second exponentially decreasing trend starts.*

Figure 7 shows how "weight loss" (y-axis) evolves with more and more iterations (x-axis) in an example instance. In this instance, we used a $1,000 \times 4M$ matrix, which corresponding to 1,000 routers each generating a bitmap of size 4Mb. We "plant" a pattern of size 100×30 into it. The heaviest 4,000 column vectors are selected as $S_1 = S$ according to Theorem 2, and the number of columns contained in the pattern and also in S_1 is 15. Therefore, our algorithm is supposed to stop at the 15th iteration, but it does not know this number in advance. We now show in this instance, our procedure will allow us to find this number 15. We can see from Figure 7 that the curve takes an exponential dive at the very beginning, then flattens out, and finally takes another exponential dive. The second exponential dive starts right after 15 iterations. Therefore, our procedure will choose 15 as the right number of iterations by the aforementioned termination criteria.

C. Non-naturally-occurring and detectable thresholds

Upon detecting a "pattern" at the central monitoring station, we need to determine if this pattern is interesting in the sense that it is not "naturally occurring". A pattern is *naturally occurring* if the probability for it to exist in the data that does not contain any common content is beyond a threshold ϵ. Detecting a naturally occurring pattern is clearly meaningless. For example, in the aligned case, the submatrix that we are searching for has to be "large enough" to "make sense". If we chose a small pattern such as 4×4 submatrix, there will be lots of them in the matrix and none of them corresponds to the discovery of common content.

In the aligned case, the *non-naturally-occurring threshold* can be computed in a straightforward way. Given an $m \times n$ matrix, that contains half 1's and half 0's, we would like to find a and b such that an $a \times b$ submatrix of all 1 will not be naturally occurring in the $m \times n$ matrix. Since there are $\binom{n}{a}\binom{m}{b}$ possible choices of submatrices, and each matrix has the probability 2^{-ab} to contain all 1's when there is no common content, we know by Markov's inequality that the probability for an $a \times b$ submatrix to naturally occur is bounded by the following probability:

$$\binom{n}{a}\binom{m}{b}2^{-ab} \qquad (1)$$

Immediately above the non-natural-occurring threshold might not be feasible in practice. For example, to detect a submatrix of size 10×10 in a 100×100 matrix, if we are willing to try all combinations of rows and columns $\binom{100}{10} \times \binom{100}{10}$, we will find all such submatrices if there are any. Due to the randomized nature of our solution, correct detection might not be guaranteed for all submatrices that are above the non-naturally-occurring threshold. However, we can numerically compute the probability of missing a submatrix of size $a \times b$. In this manner we can determine the detectability threshold, such that a submatrix larger than this threshold is missed only with probability δ, where δ is a small constant, say 10^{-3}. The detectability threshold is larger than the non-natural-occurrence threshold, implying that some submatrices quite unlikely to occur naturally are still too small to be detected computationally. The gap between these two thresholds is a function of the computational complexity of the detection algorithms, with smaller and smaller submatrices being detected as the computational effort devoted to the detection is increased.

IV. DESIGN FOR THE UNALIGNED CASE

In Section III, we reduced the problem of finding common content in the aligned case to the problem of finding submatrix of all 1's in a large matrix. However, this technique does not work for the unaligned case. This is because, if we hash the packet fragment that starts at a fixed offset in a packet, different routers will sample different fragments of the objects due to the variable prefix at the very beginning. Even if two routers witnessed a common content, with high probability two different sequences of hash values will be produced. In this section, we design a new technique for the unaligned case that solves this problem.

A. Distributed online streaming module

As discussed in Section II-D we assume that all common content is transported using a fixed packet size. Suppose the common content is transmitted over TCP, which typically use MTU segments of 576 bytes. Each packet includes a 40 byte header and a 536 byte payload. Then the common content can have 536 different starting points (0, 1, ..., 535) in the transmitted object modulo 536. If the common content in two objects seen by two routers has the same prefix length, and the two routers sample fragments at the same offset, they will obtain the same fragments. This will produce two identical sequences of hash values. In the bit locations indexed by these hash values,

```
1. Initialization
2.     Set all bits in Arrays A₁, A₂, ..., Aₖ to 0;

3. Update_arrays(pkt)
4.     for array_index := 1 to k
5.         bit_index := hash(substring(pkt.contents,
                   offset[array_index], 20));
           /* hash the 20-byte fragment from the offset
               offset[array_index] */
6.         A[array_index][bit_index] := 1;
7.     end
```

Fig. 8

OFFSETS SAMPLING ALGORITHM FOR UPDATING THE ONLINE STREAMING MODULE

the arrays corresponding to both these routers will have value 1. However, the probability that such a match happens is only $\frac{1}{536}$, if the prefix length is distributed uniformly at random in $[0, 535]$. Also, even if we are lucky to have such a match, for a common content that is split into 100 segments, we are looking at about 100 common 1's between arrays that are both 131,072 bits wide (to be justified in Section V), assuming we are using same or similar parameters as in the aligned case. The signal is too weak to be detected.

Next, we describe our solutions – *offset sampling* and *flow splitting* – to address the above two problems.

Increasing matching probability. To increase the probability that we are going to have a match, instead of each router taking a fragment from a fixed location, each router picks a set of k random offsets chosen beforehand and fixed for a measurement epoch. For each packet, the router samples a total of k fragments, starting at these offsets. The hash values will be used as indices to write into k different arrays, one array corresponding to each offset. Figure 8 shows the offset sampling algorithm.

In general, using k offsets amplifies the probability of having a match by approximately k^2. Suppose the offsets used by router 1 are $a_1, a_2, ..., a_k$ and the offsets used by router 2 are $b_1, b_2, ..., b_k$. Then $((a_i - b_j) \text{ modulo } 536)$ is a set of k^2 random numbers. Let the common content seen by these two routers have prefix length l_1 and l_2, respectively. If $((l_1 - l_2) \text{ modulo } 536)$ matches any of the $((a_i - b_j) \text{ modulo } 536)$, the fragments taken from segments of contents 1 at offset a_i will be the same as fragments taken from segments of contents 2 at offset b_j. In this case, there will be 1's in a common set of indices in both the i_{th} array of router 1 and the j_{th} array of router 2. Since, given a fixed set of a_i and b_j, there are about k^2 combinations of i and j, the probability for such a match is increased by k^2. To be accurate, this increase is slightly smaller than k^2 due to some collisions. The probability for two arrays to have such a match is actually captured by $1 - e^{\frac{-k^2}{536}}$. In general, to achieve similar matching probability, for large-size packets, we should use larger value of k. However, since the probability increases approximately quadratically with k, the value k only needs to be $\sqrt{\delta}$ times larger when the packet size is δ times larger.

In this paper, we will fix the number of arrays to 10, targeting the packet size of 536. For packets around 500 bytes in length, we will use 10 different offsets, one offset per array. For packets 1000 bytes and above, we will use 20 different offsets, two offsets per array. We will not perform such operation for packets smaller than 500 bytes, which we will justify in Section V. This effectively limits the computational and memory complexity of

```
1. Split_flow
2.     Upon the arrival of a packet pkt
3.         group_index := hash(pkt.flow_label);
4.     call Update_arrays(pkt) to update all arrays of the
5.         group indexed by group_index;
```

Fig. 9

OFFSET SAMPLING PLUS FLOW SPLITTING ALGORITHM FOR UPDATING THE ONLINE STREAMING MODULE

this operation to 10 bits per 536 bytes of traffic.

Magnifying signal strength. Now the probability of having a match is increased by k^2, but we have to solve the problem of weak correlation between two matching arrays. We have argued that it is extremely difficult to identify a match of 100 packet segments between two 4M bit arrays. To increase the signal strength, we need to reduce the size of each bit array. Therefore, we split the overall traffic into multiple group of arrays. Our scheme requires that packets belonging to the same flow go to the same array. We use a standard technique of splitting traffic into *groups* according to the hash values of their flow labels to ensure this. Figure 9 shows the flow splitting algorithm. Note that, there can be multiple instances of the same content passing through a given router. Flow splitting allows multiple instances of the same content to be registered in separate bit arrays. This further increases the signal strength.

In the following analysis, we assume that the original array size[6] is $2^{17} = 131,072$ bits (to be justified in Section V). The overall traffic contained in the big array is split into 128 groups according to the hash value of their flow labels. Each group has 10 arrays for offset sampling. This results in 1,280 arrays of 1,024 bits each. Such a split magnifies the signal strength by an order of magnitude. For example, suppose two instances of a common content are split into 100 packets each. They will result in 1's in approximately 100 common indices in the corresponding arrays of size 1,024 bits, if there is a match. Such an increase in the number of 1's in common locations can be easily detected.

However, there is a small tradeoff with such flow splitting. First, we simply have more rows in the matrix to deal with in the data analysis stage, which increases the computational complexity of the analysis algorithm. Our techniques to alleviate this can be found in Section IV-D. Second, some flows can be larger than others, and such burstiness in flow sizes can lead to more 1's in the array. However, this turns out to be a minor issue for two reasons. First, the Zipfian nature of the traffic determines that only a very small number of flows are big enough to cause this kind of problem. Each of such instances will only affect one array. Second, the fact that the measurement epoch is just one-second worth of traffic further limits this burstiness. Our measurements on tier-1 ISP traffic traces show that such burstiness causes negligible deviation to our results that assume perfectly balanced flow splits.

In summary, each router will generate a matrix of 1,024 bits in width and 1,280 in height[7]. These matrices will be shipped to the data analysis center for analysis.

---

[6]Note that this is different from the assumed array size of 4 million bits in the aligned case.

[7]The height of the matrix can be tuned depending on the speed of the router.

B. Data analysis module

Once matrices are shipped from data streaming modules, they will be merged vertically to produce a giant (in number of rows) matrix of $1,024$ columns. The function of the data analysis module is to assist in the detection of common content. We propose two "tools" in this respect. One answers the question whether there exists common content or not. The second answers the question which set of routers potentially witnessed the common content. Both tools can trigger external means such as packet logging or intrusion detection to find the common content. Both tools are very useful depending on the external means that are available to ISPs. Our first tool, based on statistical test on random graphs is extremely accurate. Our second tool is a greedy algorithm exploiting the statistical property in our bitmap construction and outputs a set of routers which with high probability have witnessed the common content.

Statistical test on random graphs. The common content detection problem in the unaligned case can be reduced to the problem of performing statistical test on a random graph $G(n, p)$. Here $G(n, p)$ denotes a random graph that has n vertices, and the events of any two vertices having an edge between them are independent and each event happens with probability p. Each bitmap corresponds to a set of vertices in this graph and correlations caused by the common content is translated into a higher probability for some vertices to have an edge between them than the "background" probability p.

Our statistical test problem is that, given a graph that contains n vertices, we would like to test whether it is an instance of Erdös-Renyi (ER) random graph $G(n, p_1)$ [6], or there exists a subset of vertices in the graph such that the probability for any two vertex to have an edge between them is larger than p_1. We refer to the latter as "preferential attachment". In statistical testing, the former is the *null hypothesis* and the latter is the *alternative hypothesis*.

We now show how we convert the matrix to an Erdös-Renyi (ER) random graph $G(n, p_1)$. Consider a $10n \times 1,024$ matrix, where the traffic is split into n groups and each group results in 10 arrays corresponding to 10 different offsets. We convert this matrix to a graph with n nodes, each node corresponds to a group. Whether an edge exists between two groups depends on the maximal number of common 1's among pairs of rows of them. The key in transforming this matrix to a random graph $G(n, p_1)$, when there is no preferential attachment, is to keep the probability of having an edge between two nodes uniform ($= p_1$). Since the number of 1's in the rows of different groups are different, we need to set different thresholds accordingly. That is, given two rows that belong to two different groups (vertices) A and B containing i and j 1's respectively, if the number of indices at which both rows have value 1's is higher than $\lambda_{i,j}$, we add an edge between A and B. We put at most one edge between any two vertices. Also we do not establish an edge from a vertex to itself. Therefore, the resulting graph is a simple graph.

Let $X(i, j)$ be the random variable denoting the number of common 1's between two rows that contain i and j 1's respectively. When there is no "matching" between them, the probability that the number of common 1's is greater than $\lambda_{i,j}$ is given by $P[X(i, j) > \lambda_{i,j}] = 1 - \sum_{k=0}^{\lambda_{i,j}} P[X(i, j) = k]$ where $X(i, j)$ follows a hyper-geometric distribution $P[X(i, j) =$

$k] = \frac{\binom{i}{k}\binom{N-i}{j-k}}{\binom{N}{j}}$ (Here the value of N is 1,024). We set a list of thresholds $\Lambda = \lambda_{i,j}, (0 \leq i, j \leq 1,024)$ in such a way that the value of $P[X(i,j) > \lambda_{i,j}] \approx p^*$ for any value of i and j, where p^* is defined by the probability of matching between two arrays. The value of p_1, the probability of the existence of an edge between two groups, can be estimated by $p_1 \simeq 1 - (1 - p^*)^{100}$. It is easy to see that, given p_1, we can find the list of corresponding threshold $\lambda_{i,j}$. This completes our transformation of a given matrix to an Erdös-Renyi random graph $G(n, p_1)$ when there is no "preferential attachment" in the matrix.

We now describe our novel technique for testing the null hypothesis on whether our constructed graph is an instance of $G(n, p_1)$ against the aforementioned alternative hypothesis. Our idea is to check whether the size of the largest connected component in the graph significantly deviates from that is typical in $G(n, p_1)$, based on the following phase transition phenomenon [6] of the Erdös-Renyi random graph. When the probability p is less than $\frac{1}{n}$ in a random graph $G(n, p)$, with high probability, all connected components are of size $O(\log n)$. However, when p is greater than $\frac{1}{n}$, a giant connected component of size $\Theta(n)$ begins to emerge. Although the theory holds as an asymptotic result when n approaches infinity, for sufficiently large n, this phase transition phenomenon will be observed. In our ER test, we tune a set of parameters to keep the expected p_1 below the phase transition threshold. If there is no "preferential attachment", the largest connected component should have an expected size q which is $O(\log n)$. However, if the graph contains significant "preferential attachment", certain pairs of nodes will be connected by an edge with probability much higher than p_1. These edges will "merge" multiple largest connected components in the original graph into a much larger connected component than q. This simple test turns out to be miraculously accurate, with very low false positive and false negative, as we will show in our evaluation.

Detection algorithm - finding the pattern. Our statistical testing algorithm determines whether there are common content in multiple rows of the matrix, but does not identify these rows. We now introduce a greedy algorithm that finds a large portion of these rows with high probability. Note that it is not necessary to find all the rows; even a subset of all such rows will offer us enough clue for identifying the common content. Unlike the aligned case, in general we will not be able to obtain the hash indices that packet segments of the common content hashed to. For example, if such common content is a propagating worm, it does not find the "hashed signature" of the worm. However, since it identifies the groups that contain the common content, we believe that information exchange at finer granularity between involved routers (the subset identified as having seen the common content) will help identify its signature, using a much smaller subset of aggregated traffic. Our procedure in finding these rows consists of three steps.

Step 1: Constructing the graph. We induce a new graph out of the matrix using a different threshold $\lambda'_{i,j}$. We work with this new graph rather than the graph used for statistical testing because the latter graph often does not offer us the best accuracy on finding the core. In fact, while the graph for statistical testing uses $\lambda_{i,j}$ that results in p_1 smaller than the phase tran-

```
1.  FindCore(G)
2.     construct G' = (V', E'), a duplicate copy of G
3.     repeat
4.        let v be the vertex in G' with the smallest degree;
5.        delete v and edges incident on v from G'
6.     until |V'| ≤ β
7.     define V_core as the remaining vertices in V'
```

Fig. 10

CORE FINDING ALGORITHM FOR THE UNALIGNED CASE

sition threshold $\frac{1}{n}$, in this new graph the p_1 induced by $\lambda'_{i,j}$ is much larger than $\frac{1}{n}$. This step is straightforward and will not be discussed in detail below.

Ideally, we would like to find the clique (or a dense subgraph such that almost every node is connected to every other node in this subgraph) of maximum size in graph G'. However, the maximum clique problem is NP-hard and can not be approximated within a factor $|V|^{1/2-\epsilon}$ for any $\epsilon > 0$ in the general case. However, our problem has nice statistical property that general polynomial-time algorithm for maximum clique can not exploit. We next use the property to complete our greedy algorithm.

Step 2: Finding the core. Figure 10 illustrates the procedure for finding the core. We first copy the original graph G to G' and perform all the operations on G' (we will need G in Step 3). We keep deleting the nodes with the smallest degree and their associated edges from the graph G, until the number of vertices in this graph becomes β. The remaining vertices are the core we are looking for, denoted as V_{core}. Through Monte-Carlo simulation, we configure β such that if the number of vertices containing the common content are beyond a detectable threshold, with high probability the majority of the vertices in the core contain the common content we are searching for.

Step 3: Using the core to find the rest. Once we find a core, we will use the core to find a large portion of the rest of vertices containing the common content. We denote the set of vertices in G that are not in the core as $\overline{V_{core}}$ and the graph they induce as $\overline{G_{core}}$. Our algorithm removes from $\overline{G_{core}}$ all nodes that have less than d edges connected to vertices in V_{core} and edges that are incident on them. We denote the resulting graph as H. We set d such that a large portion of the vertices that have the common content "survive" and the majority of the other vertices do not. Again, Monte-Carlo simulation allows us to set d properly given a set of operating parameters. This core contains a small set of vertices that witnessed the common content with very high probability. To include more vertices that witnessed the common content, we run the algorithm $FindCore$ again on the graph H to obtain a core from H, denoted as V_{2nd_core}. Our final result is the union of two cores V_{core} and V_{2nd_core}.

This process of core finding could be repeated to obtain more vertices that contain the common content. However, our experiments suggest that most of the vertices we are looking for are already in $V_{core} \bigcup V_{2nd\_core}$ and the return is marginal afterwards since the remaining graph becomes more "noisy". In other words, we will not be able to find more such vertices without causing high false positives.

Our algorithm is stochastically optimal. Our algorithm shown in Figure 10 for finding the core, is stochastically optimal, under a reasonable computational model, in the following sense. Among all the algorithms that fall under the computa-

tional model, our algorithm produces a core that has the lowest average false positive. In other words, the average number of vertices that do not belong to the core (not containing common content) is kept to the minimum by our algorithm. This result is proven using machineries from the stochastic ordering theory [7]. Please see Appendix for details of the proof.

C. Non-naturally-occurring and detectable thresholds

We mentioned before that it only makes sense to detect patterns that are not naturally occurring. The detectable threshold can be much higher than the non-naturally-occurring threshold. In the unaligned case, the methods for estimating the non-naturally-occurring threshold and the detectable threshold are quite different from the ones in the aligned case.

The problem of finding the non-naturally-occurring threshold in unaligned case can be formulated as follows. For the aforementioned $10n \times 1024$ matrix and a common content that is split into s packets, we would like to determine the minimum value for a parameter m, which is the number of nodes in a subgraph. This parameter m has to satisfy the following two properties. First, if G does not contain any common content, then there exists a threshold value d such that the probability for a subgraph of size m to have more than d edges is very small (e.g., 10^{-10}), given the probability for two random nodes to have an edge in the original graph G (p_1 as defined above). Second, if G does contain a pattern of size no less than m, for the **same** d, the probability for a subgraph of size m that is a subgraph of the pattern graph (with edge probability p_2 as discussed above) to have more than d edges is large enough (e.g., > 0.95). These two requirements correspond to both low false positive and low false negative. Using Markov's inequality, given a value d, the first probability can be bounded by

$$\binom{n}{m}(1 - binocdf(d, m(m-1)/2, p_1)) \qquad (2)$$

The second probability can be computed as

$$binocdf(d, m(m-1)/2, p_2) \qquad (3)$$

Our goal is to find a minimum m such that there exists at least one value d such that the first probability is very small and the second probability is larger than a certain threshold. In statistics, this corresponds to setting a threshold on type-I error and then trying to minimize type-II error.

To find the smallest possible m, we need to tune both p_1 and d. In other words, when we use a larger p_1, d has to be accordingly larger, and vice versa. The nature of the problem suggests that there is a sweet spot in the middle. Despite our effort, we do not have any analytical results[8] that guide the co-tuning of p_1 and d. However, we implemented an efficient numerical analysis procedures that search for the best combination of p_1 and d in a brute-force way. In Section V, we will show results produced from this procedure.

As explained before, a pattern that is barely non-naturally-occurring may not be detectable since the detection algorithm

operates under a stringent computational complexity constraint. In our context, we define a pattern to be detectable as follows. When we run the algorithm shown in Figure 10 to find a core, the pattern has to be large enough at the very beginning to produce a core of size at least β and with high probability this core contains very few false positive vertices (those that should not be in the core). This ensures that we will find enough vertices in the pattern in the third step of our algorithm.

D. Computational complexity

The vast majority of the computational complexity of both our ER test and the pattern finding scheme comes from computing, for any two rows in the matrix, the number of indices in which both row have value 1. Other algorithms, mainly manipulating the graph induced from the matrix, have a low complexity of at most $O(|E|)$ (the induced graphs are all sparse graphs such that $|E|$ is about $O(n)$). For a matrix with $10n$ rows, we need $O(100n^2)$ bitwise-AND operations. We will show below that if we would like to monitor thousands of high-speed links (e.g., OC-48), we will have an n that is in the order of 100,000. In this case, $100n^2$ is 1 trillion operations. We estimated that it will take a few hours in software implementation. However, the network generates such a workload every second!

We suggest several possible ways, used alone or in combinations, to cope with this complexity. The first possibility is that if we are willing to reduce the number of OC-48 links to be monitored together to hundreds, we automatically reduce the complexity by about 100 times. However, a large network often has much more than a few thousands ingress/egress links to monitor. It is desirable to monitor as many of them together as possible, to make the probability of detecting a weak (but potentially interesting) signal higher. The second possibility is that we can simply sample 10% of the vertices and find a core only in this subset. Then this core will be used to find other vertices in the pattern, which has $O(n)$ complexity since the core is relatively small. This reduces the complexity by about 100 times, i.e., each such computation will take a couple of minutes. The tradeoff here is that we can only detect patterns that are several times larger than detectable if we do not perform sampling. In other words, the sensitivity of the algorithm in detecting patterns goes down. The third possibility is to distribute the load to a large number of CPU's. Since this computing job has no data dependence between any two operations, it is ideal for massive parallel processing (a.k.a, embarrassingly parallel). However, a few thousand CPU's are needed for this gigantic task, the cost of which is nontrivial. The fourth possibility is that we design special hardware that can perform tens of thousands of such long (1024 bits) bitwise-AND operations in a single cycle. Once this hardware helps us generate the graph, the rest of our algorithms takes a few seconds. Finally, the fifth possibility is to sample a small percent of the measurement epochs for analysis. Hopefully the patterns will span enough epochs to be detectable even with sampling.

V. EVALUATION

A. The aligned case

We use Monte-Carlo simulation to evaluate the accuracy of our pattern detection algorithm presented in Section III. The

---

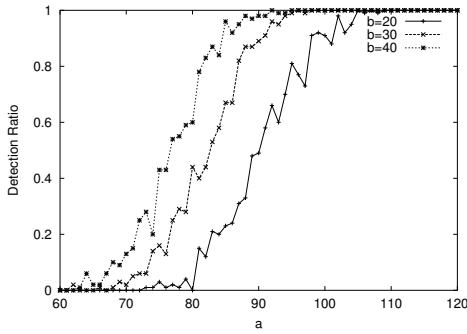[8]The dependence induced by order statistics [7] makes the analysis intractable.

Fig. 11

THE DETECTION RATIO OF THE DETECTION ALGORITHM FOR THE
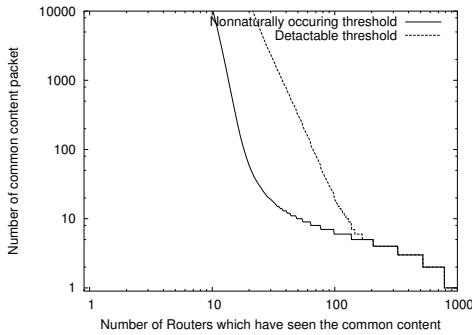ALIGNED CASE

Fig. 12

THE BOUND OF NON-NATURALLY-OCCURRING THRESHOLD AND
DETECTABLE THRESHOLD

problem size we are targeting is as follows. We would like to monitor 1,000 OC-48 (2.4 Gbps) links simultaneously. As we explained before, the number of columns in the matrix is 4M. Therefore, the matrix we are dealing with has the size 1,000 × 4M.

In our Monte-Carlo simulation, we randomly generate a 1,000 × 4M 2D bitmap in which a bit is set to 1 with probability 0.5. Then, we inject an $a \times b$ pattern by randomly selecting a rows and b columns, and setting all intersecting bits to 1's.

1) Performance of the greedy algorithm: We test how well our greedy algorithm for core-finding works. Figure 11 shows the detection ratio, which is the empirical probability of the pattern being detected, obtained from the Monte-Carlo simulation. Note this value is equal to 1 minus the false negative ratio. We do not simulate the false positive ratio here since it is extremely small from the theoretical analysis. Our simulation is repeated 100 times and the values shown in Figure 11 are the average of the 100 simulations. The three curves in Figure 11 correspond to a pattern that has 20, 30, and 40 rows (routers) respectively. The x-axis is the number of rows (routers) and the y-axis is the detection ratio. These three curves clearly show that the detection ratio becomes larger when the pattern size becomes larger. For example, for the target size of 100×30 discussed above, we can successfully detect the pattern with probability close to 0.988. In other words, if there is a large enough core that survives the "screening by weight", our greedy algorithm will almost never miss it.

2) Non-naturally-occurring and Detectable thresholds:
The non-naturally-occurring threshold curve (lower one) for the 1000 × 4M matrix is shown in Figure 12. The x-axis and y-axis of the figure correspond to the row size a and column size b of the all-1 submatrix, respectively. Recall that a corresponds to the number of routers that have seen the common content and b corresponds to the size of the common content in number of packets. Areas lower than (and left to) the non-naturally-occurring curve represents $a \times b$ values that are naturally occurring, and vice versa. Clearly there is an intuitive tradeoff between a and b. For example, when a is 28, b has to be at least 21 for the pattern to be non-naturally-occurring. On the other hand, when a becomes 70, b only needs to be no less than 10.

Also shown in Figure 12 is the detectable threshold curves. Recall that our algorithm first finds a core among the heaviest 4,000 columns, and then uses it to find (most of) the rest of the columns. We plot the curve in such a way that any pattern of size combination ($a \times b$) above the curve can be detected with at least 95% probability. Note that the detectable threshold curve always lies above the non-naturally-occurring threshold curve, representing the unfortunate tradeoff between detectability and the computational complexity (running an quadratic algorithm on 4,000 columns rather than on 4,000,000 columns). Let us use the same example a values as above and compare the corresponding b values. When a is 25, b needs to be at least 3,029, which is two orders of magnitude larger than 21. This is because the signal of 25 1's in an array of 1,000 bits is so weak that it has to be "amplified" by 3,029 times! When a becomes 70, the value of b only needs to be 99. But this value is still about 10 times larger than the non-naturally-occurring threshold (i.e., 10) with the same a value.

We briefly explain the procedure for estimating the aforementioned detectable thresholds using one point on the curve, $(100, 30)$, as an example. Then, consider how many columns will be selected when we use 550 as the threshold for the weight, the number of 1's in a column. In a column that does not correspond to a packet in the common content, the probability that there are more than 550 1's in this column is $1 - binocdf(550, 1000, 0.5) \approx 0.00073$. There will be on the average $4M \times 0.00073 \approx 2900$ columns that are heavier than 550 and that do not contain common content packets. Since $2,900 << 4,000$, by large deviation theory, the probability for the number of such columns to be close to 4,000 (thereby squeezing out the pattern) is very low. Now in a column that corresponds to a packet of the common content, the probability that its weight is over 550 is about 0.55. So if there are 30 packets that belong to the common content, then probability that there at at least 8 packets that survive the "screening by weight" is $1 - binocdf(7, 30, 0.55) = 0.988$. Note that we choose 8 because the pattern of 30×8 is the non-naturally-occurring threshold in the matrix consisting of the heaviest 4,000 columns. Therefore, we have a detection probability of about 0.988.

Our Monte-Carlo simulation to evaluate the performance of the greedy algorithm for several points on the detectability curve shown in Figure 12 shows that the algorithm works 100% of time in all cases. Note that there is a positive probability for this greedy algorithm to fail to find the core. However, this

probability is simply too small to be reflected in thousands of repeated trials.

B. The unaligned case

Our mechanism is designed to monitor thousands of OC-48 (2.4 Gbps) links simultaneously. In the unaligned case, such monitoring becomes much more difficult compared to the aligned case. We simplify the problem slightly as follows[9]. We assume a minimum packet size of 4,000 bits because the streaming algorithm will not perform operations on packets shorter than 500 bytes. We also assume that we are only monitoring no more than $\frac{1}{8}$ of the traffic (i.e., about 75,000 packets from an OC-48 link in each one second measurement epoch by ignoring flows that are very large (i.e., elephants)[10]). Since putting $(\ln 2)l$ random bits [4] into an l bit array will make the array contains approximately half 0's and half 1's, an array of 131,072 bits will do. With each row of size 1,024 bits, the traffic needs to be split into 128 groups, generating 1,280 arrays (rows) with 10 different offsets. Monitoring 800 such links will result in 1,024,000 rows in the matrix, which will induce a graph with 102,400 vertices. Hereafter, the matrix we are dealing with is $10n \times 1024$, where n is 102,400.

1) Erdös-Renyi test: In Section IV, we have discussed the mechanism of using Erdös-Renyi test to determine the existence of a common content pattern. In this section, we study the sensitivity of Erdös-Renyi test in terms of false positive and false negative probabilities using Monte-Carlo simulations. The *false positive probability* is the probability that a random graph is misinterpreted as a graph with common content pattern. The *false negative probability* is the probability that a graph with common content pattern is considered to be a random graph.

We assume that the common content is packetized into 100 packets. We compute the aforementioned threshold table Λ using $p_1 = 0.65/10^5$. Note that the phase transition probability for an ER random graph of this size is $1.024/10^5$ which is larger than p_1. Then, we run Monte-Carlo simulations by varying the number of vertices n_1 that have seen the common pattern.

Figure 13 shows the cumulative distribution of the size of the largest connected component, for random graphs and graphs that have seen a pattern. We observe that the larger the number of vertices that have seen the pattern, the larger "distance" between these two CDF curves. If we set the threshold of the largest component to the same number (i.e., 100), there is almost no false positive in deciding the existence of a pattern in all three cases of n_1. However, we observe some false negative cases. The corresponding false negative probabilities are 16.6%, 5.2%, and 1.0% for n_1 = 120, 130, and 140, respectively. Note that some false negative are tolerable since such detection is performed every second. Even if the pattern is missed in one second, it may be caught in the following seconds. We can select the size of the largest connected component which strikes a balance between false positive and false negative probabilities. We also observe that, a pattern of a small number of

---

[9] This simplification will also work for aligned case. However, it is not necessary in the aligned case because its computational complexity is managable.

[10] Elephants can be analyzed locally in a very efficient manner using [8], for example. Our goal here is mainly to detect "a group of mice"; detecting a group of elephants is clearly an easier problem sidestepping which allows us to focus our computation and storage resource on the more interesting problem.
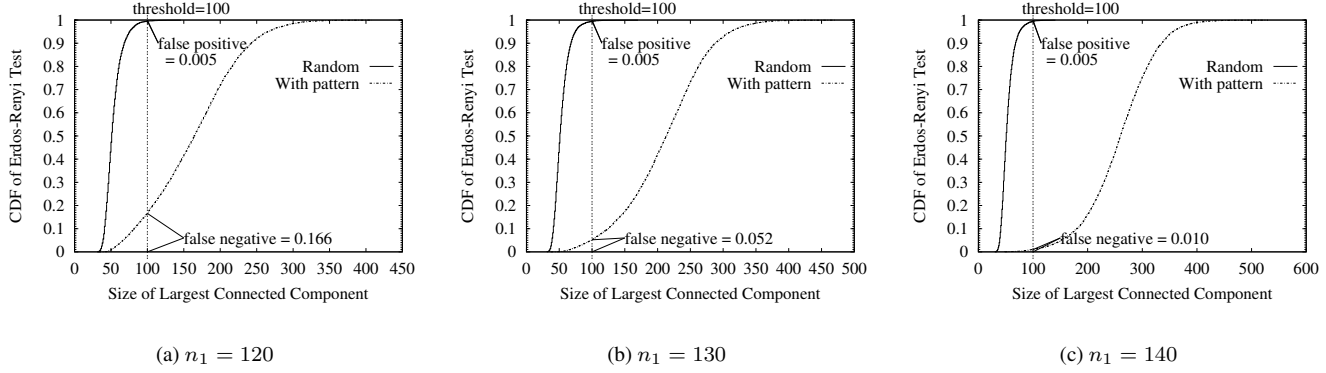
# packets in common content	n_1	Average core size	Average false negative	Average false positive
100	125	65.3	0.485	0.014
	144	112.1	0.241	0.025
	165	154.4	0.099	0.037
110	67	35.6	0.481	0.023
	77	59.3	0.239	0.012
	89	81.8	0.096	0.017
120	44	22.4	0.491	0.001
	51	38.5	0.249	0.006
	57	51.9	0.092	0.002

TABLE I

AVERAGE SIZE OF CORES DETECTED BY GREEDY ALGORITHM

nodes—n_1 correlated vertices out of $102,400$ vertices—is very effective in connecting the smaller connected components in forming a rather larger one. This shows that the Erdös-Renyi test is very sensitive in detecting non-naturally occurring patterns.

2) Finding the core using Monte-Carlo simulation: Table I shows the average size of cores detected by our greedy algorithm. Here we set the value of p_1 as $0.8/10^4$, which is higher value than the one for Erdös-Renyi test (as explained in Section IV-B), and compute the corresponding threshold table Λ'. Given the number of packets in common content in the first column of the table, three values of n_1 in each line shows the minimum value of n_1 to make the average core sizes more than 50%, 75%, and 90% of n_1, respectively. The third and fourth columns show the average false negatives and false positives in the detected core. The probabilities of false positives and false negatives was previously used in evaluating the performance of the statistical test on random graphs that operated on the entire graph to return a binary (yes/no) answer to the question of whether or not a pattern existed in the graph. The detection algorithm, on the other hand, tries to identify the individual routers that have seen instances of the common content. Therefore, the definitions of false positive and false negative are different. A false positive in this case, corresponds to a router being mistakenly identified as having seen the common content, and a false negative correspond to routers that have seen the common content being missed by the detection algorithm.

There is a clear tradeoff between the size of the common content and the number of vertices that need to contain it to be statistically significant. For example, when there are 100 packets in the common content and 125 vertices in the pattern, the greedy algorithm will find a core of average size 65.3 (out of 125), which contains 51.5% of the vertices in the pattern. With 144 and 165 vertices, we can increase the size of the core to 75.9% and 90.1% respectively. If the number of packets in the common content increases to 120, we only need 44, 51, and 57 vertices to get 50%, 75%, and 91% of the cores, respectively. In all simulation results, we get very small false positive values.

3) Non-naturally-occurring and detectable thresholds: Table II shows the size of the non-natural-occurring pattern size in a graph of 102,400 vertices. For example, if the common content consists of 100 packets, then a minimum of 95 vertices should have the common content for the resulting statistical pattern to be meaningful. This size is obtained using the aforementioned numerical procedure to co-tune the parameters m and d

(a) $n_1 = 120$ (b) $n_1 = 130$ (c) $n_1 = 140$

Fig. 13

THE EFFECT OF n_1 ON THE FALSE POSITIVE AND FALSE NEGATIVE PROBABILITIES IN ERDÖS-RENYI TEST

Common content in packets g	Minimum Size of m
80	297
90	150
100	95
110	62
120	46
130	36
140	28
150	23

TABLE II

EXAMPLES OF BOUND OF NON-NATURALLY OCCURRING CLUSTER

Common content in packets g	Detectable threshold m	Average core size
100	150	56
125	80	50
150	50	30

TABLE III

SET OF THRESHOLDS

(the number of edges d in a subgraph with m vertices). Similar to the aligned case, there is a clear tradeoff between the size of the common content and the number of vertices that have to contain it to be statistically significant.

Table III shows the detectable thresholds that can be achieved by our greedy algorithm. For example, when there are 100 packets and 150 vertices in the pattern, the greedy algorithm will find a core of average size 56. This core size is decent enough for us to find a large portion of other vertices in the pattern. By comparing numbers in Tables II and III, one can notice that the detectable pattern size is always larger than the non-naturally-occurring threshold.

4) Stress test using tier-1 ISP trace: In this section, we evaluate our algorithms using Internet packet header traces. If the traffic is evenly split into different groups according to the hash values of the flow labels, the result should be identical to our analytical and Monte-Carlo simulation results. However, due to the burstiness of the traffic, some groups will have more packets hashed to it and some will have less. We would like to evaluate the impact of this burstiness on the robustness of our greedy algorithm.

The Internet packet header trace we used in our experiments was collected at an outgoing link that connects a data center to a tier-1 ISP's backbone network. The trace contains a total of 150 million packets, and the traffic load was very bursty.

We generate the 2D-bitmap corresponding to this "bursty traffic" as follows. Because we do not have traffic traces from multiple interfaces, we use the traffic segments from the same trace in different epochs to simulate that. The trace is cut into segments of certain number of packets each. Each segment corresponds approximately to one second worth of traffic. Then we split each segment into 32 groups according to the hash value of the flow label. Each group consists of 10 arrays, corresponding to 10 offsets. A packet that is hashed to a group results in one bit being put into a random position in each of the 10 arrays. This captures the burstiness of flow splitting in real-world traffic. In this way, one segment of traffic fills up 320 arrays of size 1,024 to 50%. The next segment will be used to fill 320 new arrays.

After we generate a 2D-bitmap of size $1,024,000 \times 1,024$, we insert patterns of various sizes into it. Since we use the hash function to generate the content signature, the performance of our algorithm can be constrained by the randomness of the hash function and/or the input traffic. We assume that we can use the fast and efficient hardware implementations of hash function such as [9]. And our randomness test for the input traffic shows that the traffic has almost random value of the contents. we believe that the effect of the hash collision is negligible.

We found that the detectable threshold using this "bursty traffic bitmap" is slightly lower than that obtained through Monte-Carlo simulation assuming even traffic distribution. For example, to find more than 50% of core when there are 100 packets in the common content, we need about 121 vertices in the pattern. In comparison, with the same parameters, our Monte-Carlo simulation suggests that 125 vertices in the pattern are needed. Clearly, in this case the burstiness comes to our advantage. This is because, due to the Zipfian nature of the Internet traffic [10], a small number of rows that large flows (small in number) are mapped to, absorb a large percentage of traffic, so that the other rows become very lightly loaded. Although signals contained in the rows that large flows are mapped to are mostly lost, signals contained in other rows, which is the vast majority, are amplified.

VI. RELATED WORK

The most closely related works to ours are [11], [12] in theoretical computer science and database communities. In [11], Feigenbaum and Kannan proposed to ship "synopses" of the raw data from physically separated network elements to a central operations facility. They presented a space-efficient one pass algorithm to compute the L_1 differences between two data streams. The method proposed in [11] can be considered as constructing families of limited-independence random variables that are range-summable, that is the sum is computable in polynomial time. In [12], Babcock and Olston proposed techniques to answer top-k queries for values continuously updated from distributed monitoring stations by compensating the local skew with factors that make the local top-k appears as the global top-k values. This reduces the update that needed to send to the central station. However, none of these techniques are applicable in our context.

Previous work in the networking literature has focused on obtaining relevant traffic characteristics using Bloom Filter or synopsis data structure from a single router or link [13], [14]. For example, Estan and Varghese [8] presented algorithms that can identify and monitor a small number of elephants with a small amount of fast memory. Algorithms to detect significant flow-size changes have also been proposed [13], [15], [16]. However, these techniques can not be applied in our common content detection problem since the "signal" in each local station is too weak to be detected. We argue that our problem inherently require a distributed collaborative data streaming approach.

Techniques exploiting properties of specific common content such as worm have been proposed. These techniques assume a single observation or vantage point [17], [18], [19]. For example, Singh et al. [17] have proposed the EarlyBird system for automated worm fingerprinting. They use Rabin fingerprints over a sample of substrings in the packet payload to update a content prevalence table. When used in conjunction with counts of source and destination IPs for packets carrying the corresponding content, a comprehensive worm detection solution can be built. Our work extends the scope of common content detection techniques using the DCS model, to enable distributed versions of such single vantage-point solutions. The techniques required to detect common content from multiple vantage point with high traffic volume is quite different from theirs. With the ability to monitor a significantly larger amount of traffic in a distributed manner, we believe that our common content detection techniques can be used in the next generation of such detection devices.

There have been studies on identifying similar contents [20], [21]. Broder et al. [21] proposed similarity metrics to characterize similar contents. Spring and Wetherall [20] used Rabin fingerprint [22] to compute represent fingerprints of each packet. For redundant traffic between two cache proxies, only fingerprints are transmitted to reduce bandwidth consumption. The contents is reconstructed at the other end by searching the cache indexed by the fingerprints. The bitmap scheme proposed in this paper can be viewed as simple fingerprinting solutions. These similarity metrics and fingerprinting techniques require more computation and more storage. For future work,

we would like to investigate identifying similar contents witnessed by multiple routers in the network.

VII. CONCLUSION

In this paper, we propose a set of novel data streaming techniques detecting common content in the Internet traffic based on the digests shipped back to the operation center with three order of magnitude data reduction from the raw traffic. Our algorithms exploit the fact that the common content signal is hidden in the background of random noise. We rigorously formulate our detection problem and present efficient algorithms to detect aligned and unaligned common content. Our algorithms are shown to be very effective through extensive simulations and experiments on traffic traces collected from a tier-1 ISP.

REFERENCES

[1] D. S. Hochbaum, "Approximating clique and biclique problems," *J. Algorithms*, vol. 29, no. 1, pp. 174–200, 1998.

[2] D. S. Hochbaum, *Approximation algorithms for NP-hard problems*, PWS Publishing Co., 1997.

[3] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, "Packet-level traffic measurements from the sprint ip backbone," *IEEE Network*, 2003.

[4] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *CACM*, vol. 13, no. 7, pp. 422–426, 1970.

[5] R. Peeters, "The maximum-edge biclique problem is NP complete," Research Memorandum 789, Faculty of Economics and Business Administration, Tilberg University, 2000.

[6] B. Bollobas, *Random Graphs*, Cambridge University Press, 2001.

[7] H.A.David and H.N.Nagaraja, *Order Statistics*, A Wiley-Interscience Publication, 2003.

[8] C. Estan and G. Varghese, "New directions in traffic measurement and accounting," in *Proc. ACM SIGCOMM*, Aug. 2002.

[9] M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," *IEEE Trans. on Computers*, vol. 46, no. 12, pp. 1378–1381, Dec. 1997.

[10] W. Fang and L. Peterson, "Inter-as traffic patterns and their implications," in *Proc. IEEE Global Internet Symposium*, Dec. 1999.

[11] J. Feigenbaum and S. Kannan, "Streaming algorithms for distributed, massive data sets," in *Proc. IEEE Symposium on Foundations of Computer Science*, 1999.

[12] B. Babcock and C. Olston, "Distributed top-k monitoring," in *Proc. ACM SIGMOD Conference on Management of Data*, 2003.

[13] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen, "Sketch-based change detection: methods, evaluation, and applications," in *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement (IMC)*, 2003, pp. 234–247.

[14] A. Kumar, J. Xu, J. Wang, O. Spatschek, and L. Li, "Space-Code Bloom Filter for Efficient per-flow Traffic Measurement," in *Proc. IEEE INFOCOM*, Mar. 2004.

[15] G. Cormode and S. Muthukrishnan, "What's new: Finding significant differences in network data streams," in *Proc. IEEE Infocom*, 2004, pp. 1534–1545.

[16] R. Schweller, A. Gupta, E. Parsons, and Y. Chen, "Reversible sketches for efficient and accurate change detection over network data streams," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement(IMC)*, 2004, pp. 207–212.

[17] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proc. Operating Systems Design and Implementation(OSDI)*, Dec. 2004.

[18] H. A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proc. 13th Usenix Security Symposium*, 2004.

[19] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proc. IEEE Symposium on Security and Privacy*, 2005.

[20] N. T. Spring and D. Wetherall, "A protocol-independent technique for eliminating redundant network traffic," in *Proc. ACM SIGCOMM*, 2000, pp. 87–95.

[21] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *J. Comput. Syst. Sci*, vol. 60, no. 3, pp. 630–659, 2000.

[22] M. O. Rabin, "Fingerprinting by random polynomials," Tech. Rep. TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[23] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

VIII. Appendix

1. Computational model

One can imagine that the following stochastic game between two players A and B on a random graph G is played. The graph G is generated according to a probability distribution that is agreed upon by both A and B. At the very beginning, the degree of all the nodes is shown to B, and this is all the information that B knows. In particular, B is not allowed to peek into the internal structure such as edge connectivity inside the graph. B now needs to make a decision on which vertex to delete. Once this vertex is deleted, A will reveal to B the degree of all remaining nodes after the deletion. B then has to choose another vertex to delete, and so on, until there are no more than β vertices in the graph.

The stochastic rule of this game is closely related to the Principle of Deferred Decisions (PDD) [23]. The idea of PDD in this context is that the entire set of random choices are not made in advance. Rather, at each step of the game, A fixes only the random choices that must be revealed to B. At the very beginning, A only commits to the fact that the graph G eventually revealed to B at the end has the degree sequence that has been revealed to B. Once a node v is deleted by B, A has to decide which set of vertices in the remaining graph that v should connect to. Note that this is a random decision that needs to conform to the aforementioned probability distribution. The probability distribution on the set of vertices that v has an edge with should be identical to the distribution of such a set conditioned on all that B has known (the degree values of the vertices). In other words, A is not allowed to lie in the probabilistic sense.

We claim in the next section that if the random graph G is induced from the matrix using a threshold λ and there is common content in some rows of this matrix, then the greedy algorithm (removing nodes with the smallest degree) is the stochastically optimal game strategy B can play in finding the core. Despite the intuitive nature of this result, its proof is nontrivial.

One may feel that this computational model may be a little restrictive, since in our problem, we can indeed see the internal structure of the graph at the very beginning. Our justification for this model is as follows. Our algorithm needs to operate under a computational complexity constraint ($O(E)$ as discussed in the next section). Although we can "see" the structure, under such a complexity constraint, we cannot "see through" it, in the sense that we will not be able to learn enough from the structure to help us pick a better choice than if we had not seen it.

2. Proof of the stochastic optimality

Definition 1: A random variable X is said to be stochastically larger than a random variable Y, written $X \geq_{st} Y$, if $Pr[X > a] \geq Pr[Y > a]$ for all real number a.

Let v_1, v_2, ..., v_l be vertices in the graph G that contain the common content. Among them, we pick an arbitrary one and denote it as u. Let $d_u(t, B)$ be the degree of u after t deletions by an algorithm B. If unfortunately u is picked by B for removal at time t', we define $d_u(t, B) = -1$ for all $t \geq t'$. Note that $d_u(t, B)$ is a *stochastic* sequence. We denote as C the greedy algorithm.

Theorem 3: For any algorithm B that conforms to the aforementioned computational model, $d_u(t, C) \geq_{st} d_u(t, B)$ for all t.

Its proof is very involved and is omitted here due to the lack of space. This theorem implies the following important corollary, which is the main result that we would like to prove. Let $N(t, B)$ be the number of vertices among v_1, v_2, ..., v_l that "survive" after t deletions by an algorithm B. One can easily verify that $N(t, B) = \sum_{i=1}^{l} 1_{d_u(t,B) \geq 0}$.

Corollary 4: $E[N(t, C)] \geq E[N(t, B)]$ for any t.

Proof: For $u = v_1, v_2, ..., v_l$, We know that $d_u(t, C) \geq_{st} d_u(t, B)$. By DEFINITION 1, we know that $Pr[d_u(t, C) \geq 0] \geq Pr[d_u(t, B) \geq 0]$. Finally by the linearity of expectation, $E[N(t, C)] = \sum_{i=1}^{l} E[1_{d_u(t,C) \geq 0}] = \sum_{i=1}^{l} Pr[d_u(t, C) \geq 0] \geq \sum_{i=1}^{l} Pr[d_u(t, B) \geq 0] = E[N(t, C)]$. ∎