

# 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 data streaming 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.*

## 1. 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 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.

### 1.1. 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.

---

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

**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 content<sup>1</sup>. 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 instance; 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.

## 1.2. Paper outline

The rest of this paper is organized as follows. In Section 2, 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 3. Evaluation of algorithms using simulations and experiments on traffic traces collected from a tier-1 ISP is presented in Section 4. Finally, we present related work in Section 5 and conclude in Section 6.

## 2. Overview

In this section, we present 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.

<sup>1</sup>Our solution cannot detect a polymorphic worm which changes its binary content during infection/propagation.

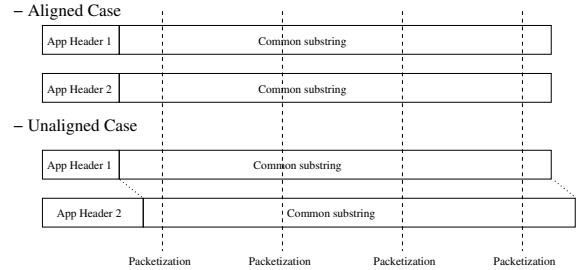


Figure 1. The aligned and unaligned cases.

### 2.1. 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.

**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.

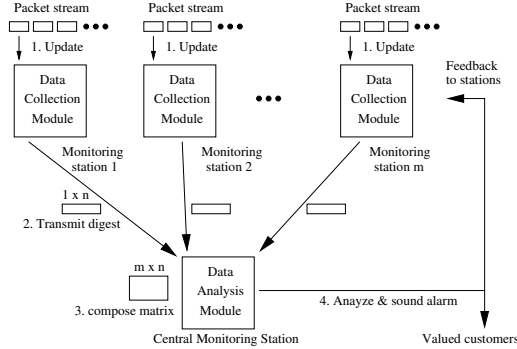
In the observed traffic flows, an object is packetized into one or more IP packets. If these packets are of a fixed size except for the last packet and this size is the same for two identical objects  $A$  and  $B$ , the  $i_{th}$  packet of  $A$  will have the same payload as the  $i_{th}$  packet of  $B$  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.

In this paper, we focus on the unaligned case which is more general and challenging problem than the aligned case. We provide, however, an overview of the mechanism for aligned case in Section 2.3 to help understanding of the mechanism for unaligned case. The detail of the aligned case can be found in our technical report [16].

### 2.2. 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 prohibitively high cost of shipping all the data to a data processing center. 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



**Figure 2. Solution framework.**

be shipped to a central analysis module for further processing. The data collection modules use specially designed data-streaming algorithms to produce succinct digests that are several orders of magnitude smaller than the 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).

### 2.3. Algorithm overview

In our schemes, 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.

In an aligned case, the data collection modules at participating routers use 4 million-bit bitmap, enough for 1 second's

traffic on an OC-48 link, to succinctly collect the signatures of the strings seen in the actual traffic. The bitmap is set to all 0's at the beginning of a measurement epoch. When a packet arrives, we hash the application layer data or a part of it to produce an index into the bitmap, and the corresponding bit is set to 1. The analysis algorithm then collect and compose bitmaps from the distributed monitoring points for analyzing. Because the common content consisting of  $b$  packets that is seen by  $a$  routers will form the same nearly  $b$  positions of 1's in  $a$  bitmaps, finding submatrix of all 1's in a large matrix can be used to find the existence of big common contents. This problem can be reduced to the problem of finding submatrix of all 1's in a large matrix, which is NP-hard in general<sup>2</sup>. Fortunately, in our special setting, the input of the problem is not arbitrary, but 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 an 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 3.1.

Detecting correlations among such complex bitmaps is also less straightforward. In section 3.2, we design a novel technique which is based on the phase transition theory of Erdős-Renyi random graphs<sup>3</sup> to detect such correlations. The phase transition theory for Erdős-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 theory 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, the size of the largest connected components in the graph becomes much larger than that should happen in the Erdős-Renyi random graph, indicating the presence of common content. This is due to the high correlation between bitmaps collected at any two nodes that have both recorded the passage of the same piece of content. As we show later, this simple test turns out to be extremely accurate.

Once this Erdős-Renyi test (described in detail in Sec-

<sup>2</sup>refer [16] for proof and the detail of the algorithm for aligned case

<sup>3</sup>Erdős-Renyi random graph used in this paper is a random graph  $G(n, p)$  that have  $n$  vertices and each possible edge independently with probability  $p$

tion 3.2) 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 [10]. 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.

## 2.4. 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 [8] 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 sizes. We remark that, our algorithms can be made to 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 that can be used on top of our algorithm to report multiple common content occurring within the same measurement epoch.

## 3. Design for the Unaligned Case

In this section, we design a technique for detecting common content for the unaligned case, and discuss the complexity of the algorithms.

### 3.1. Distributed online streaming module

As discussed in Section 2.4 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

```

1. Initialization
2.   Set all bits in Arrays  $A_1, A_2, \dots, A_k$  to 0;

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

```

**Figure 3. Offsets sampling algorithm for updating the online streaming module**

of hash values. In the bit locations indexed by these hash values, 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 4), assuming we are using same or similar parameters as in the aligned case [16]. 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 3 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  $a_1, a_2, \dots, a_k$  and the offsets used by router 2 are  $b_1, b_2, \dots, b_k$ . 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  $1 - e^{-\frac{k^2}{536}}$ . 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  $\delta$  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 4. This effectively limits the computational and memory complexity of this operation to 10 bits per 536 bytes 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;

```

**Figure 4. Offset sampling plus flow splitting algorithm for updating the online streaming module**

traffic.

**Magnifying signal strength.** Now the probability of having a match is increased by around  $k^2$ , 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 4 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 summary, each router will generate a matrix of 1,024 bits in width and 1,280 in height<sup>4</sup>. These matrices will be shipped to the data analysis center for analysis.

### 3.2. Data analysis module

Once matrices are shipped from the 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. Our first tool answers the question whether there exists common content or not. The second tool answers the question which set of routers potentially witnessed the common content. Once the existence of the common content has been found with the first tool, we may be able to use the second tool to identify the part of the routers related. Then the external means, such as packet logging or intrusion detection available to ISPs, may be used to find 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 Erdős-Renyi (ER) random graph  $G(n, p_1)$  [3], 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 Erdős-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 hypergeometric 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  $\lambda_{i,j}$ .

We now describe our novel technique for testing the null hypothesis 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 [3] of the Erdős-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. 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

<sup>4</sup>The height of the matrix can be tuned depending on the speed of the router.

- |    |   |
|----|---|
| 1. | <b>FindCore(G)</b>                                      |
| 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'  \leq \beta$                                 |
| 7. | define $V_{core}$ as the remaining vertices in $V'$     |

**Figure 5. Core finding algorithm for the unaligned case**

connected component than  $q$ . This simple test turns out to be extremely 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.

*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 transition 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 cannot be approximated within a factor  $|n|^{1/2-\epsilon}$  for any  $\epsilon > 0$  in the general case. However, our problem has nice statistical property that general polynomial-time algorithm for maximum clique cannot exploit. We next use the property to complete our greedy algorithm.

*Step 2: Finding the core.* Figure 5 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  $\beta$ . The remaining vertices are the core we are looking for, denoted as  $V_{core}$ . Through Monte-Carlo simulation, we configure  $\beta$  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.

**Our algorithm is stochastically optimal.** Our algorithm shown in Figure 5 for finding the core, is stochastically optimal, under a reasonable computational model, in the following sense. Among all the algorithms that fall under the computational 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 [9]. Detailed proof is provided in [16] and omitted here due to the lack of space.

### 3.3. 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 may generate 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.

## 4. Evaluation

Our mechanism is designed to monitor thousands of OC-48 (2.4 Gbps) links simultaneously. We simplify the problem as follows to make its computational complexity manageable. 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)<sup>5</sup>). Since putting  $(\ln 2)l$  random bits [2] into an  $l$  bit array will make

<sup>5</sup> Elephants can be analyzed locally in a very efficient manner using [5], for example. Our goal here is mainly to detect "a group of mice"; detecting a

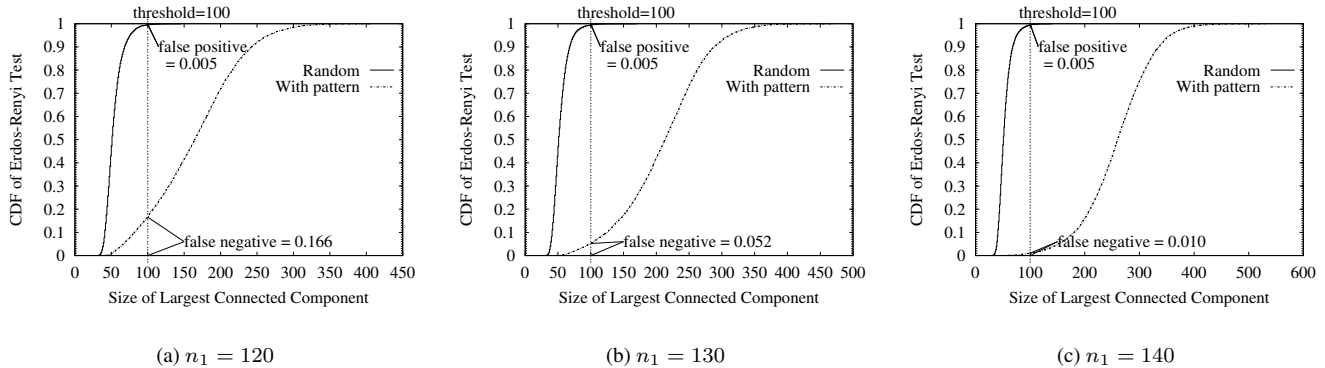


Figure 6. The effect of  $n_1$  on the false positive and false negative probabilities in Erdős-Renyi test

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.

#### 4.1. Erdős-Renyi test

In this section, we study the sensitivity of Erdős-Renyi test described in Section 3 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  $\Lambda$  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  $n_1$ , the number of vertices that have seen the common pattern.

Figure 6 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 nodes— $n_1$  correlated vertices out of 102, 400

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 1. Average size of cores detected by greedy algorithm

vertices—is very effective in connecting the smaller connected components in forming a rather larger one. This shows that the Erdős-Renyi test is very sensitive in detecting the patterns.

#### 4.2. 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 Erdős-Renyi test (as explained in Section 3.2), and compute the corresponding threshold table  $\Lambda'$ . 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 in terms of the set of routers identified<sup>6</sup>

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

<sup>6</sup> The definitions of false positive and false negative are different from the ones in Erdős-Renyi test. 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.

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.

### 4.3. 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. To generate the 2D-bitmap corresponding to this "bursty traffic", we use the traffic segments from the same trace in different epochs to simulate the traffic from multiple interfaces because we do not have the actual traffic traces from multiple points. After we generate a 2D-bitmap of size  $1,024,000 \times 1,024$ , we insert patterns of various sizes into it to evaluate our detection algorithms. Refer [16] for more detailed description of this procedure.

We found that the detectability 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 [6], 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.

## 5. Related Work

The closely related works to ours in theoretical computer science and database communities are [7, 1]. In [7], 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. In [1], 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.

In the networking literature, many techniques using a single observation or vantage point are proposed. In [12, 13], obtaining relevant traffic characteristics using Bloom Filter or

synopsis data structure have been focused. In [15, 11, 14], techniques exploiting properties of specific common content such as worm have been proposed. For example, Singh et al. [15] have proposed the EarlyBird system for automated worm fingerprinting. However, these techniques cannot be applied in our common content detection problem since the "signal" in each local station can be too weak to be detected. Our techniques required to detect common content from multiple vantage point with high traffic volume is quite different from theirs.

Gigascop [4] is a stream database designed for distributed network measurement and monitoring, which is capable of processing general database-style queries to the network data stream. However, the problem of detecting common content is a specialized functionality that is well beyond this capability.

## 6. 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] B. Babcock and C. Olston. Distributed top- $k$  monitoring. In *Proc. ACM SIGMOD*, 2003.
- [2] B. Bloom. Space/time trade-offs in hash coding with allowable errors. *CACM*, 13(7):422–426, 1970.
- [3] B. Bollobas. *Random Graphs*. Cambridge University Press, 2001.
- [4] C. Cranor, T. Johnson, and O. Spatscheck. Gigascop: a stream database for network applications. In *Proc. ACM SIGMOD*, Jun 2003.
- [5] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proc. ACM SIGCOMM*, Aug. 2002.
- [6] W. Fang and L. Peterson. Inter-as traffic patterns and their implications. In *Proc. IEEE Global Internet Symposium*, Dec. 1999.
- [7] J. Feigenbaum and S. Kannan. Streaming algorithms for distributed, massive data sets. In *Proc. IEEE Symposium on Foundations of Computer Science*, 1999.
- [8] 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.
- [9] H.A.David and H.N.Nagaraja. *Order Statistics*. A Wiley-Interscience Publication, 2003.
- [10] D. S. Hochbaum. *Approximation algorithms for NP-hard problems*. PWS Publishing Co., 1997.
- [11] H. A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proc. 13th Usenix Security Symposium*, 2004.
- [12] B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM IMC*, pages 234–247, 2003.
- [13] A. Kumar, J. Xu, J. Wang, O. Spatscheck, and L. Li. Space-Code Bloom Filter for Efficient per-flow Traffic Measurement. In *Proc. IEEE INFOCOM*, Mar. 2004.
- [14] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proc. IEEE Symposium on Security and Privacy*, 2005.
- [15] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proc. Operating Systems Design and Implementation (OSDI)*, Dec. 2004.
- [16] M. Sung, A. Kumar, L. Li, J. Wang, and J. Xu. Scalable and efficient data streaming algorithms for detecting common contents in internet traffic. Technical Report GIT-CC-06-04, College of Computing, Georgia Institute of Technology, Jan. 2006.