Rapid and Efficient Detection of Distributed Anomalous Aggregates

Ankur Jain & David Wetherall
Computer Science and Engineering
University of Washington Seattle
{ankur,djw}@cs.washington.edu

Abstract

We consider the problem of how to monitor the behavior of a large-scale distributed system to quickly and efficiently determine when any of its aggregate traffic flows exceeds a fixed rate-limit. Existing algorithms such as periodic distributed queries are either expensive or slow to detect violations. We present the design and evaluation of CoCop, a family of new, lighweight mechanisms that tie the computation of an aggregate value to the activity level of the aggregate in the system. We analyze CoCop and evaluate it with trace-driven simulations using PlanetLab flow data. We find that it can detect aggregate anomalies quickly and with orders of magnitude lower communication overhead than periodic queries in the common case of normal system behavior. Moreover, CoCop is difficult for an adversary to undermine because its efficiency is not strongly tied to the distribution of the aggregate across the system.

1 Introduction

As more open, distributed systems such as PlanetLab are deployed, there is a greater need for mechanisms that monitor the overall health of systems. We give three motivating examples that require the aggregate behavior of a distributed system to be checked for anomalies. First, Internet measurement experiments are now routinely run across hundreds of Scriptroute [33] servers that probe the Internet. Scriptroute aims to ensure that the servers do not collectively overwhelm any of the IP addresses that they probe, even when multiple experiments run concurrently. Second, public DHT storage systems such as OpenDHT [17] share soft-storage across a wide pool of users and machines. To prevent misuse or abuse, the system may wish to limit allocations to a user if the storage that is charged to their IP address exceeds a specified proportion of the total storage. Third, a distributed IDS may use correlated activity at different locations, exhibited when many nodes experience a small rate of a specific event, to detect Internet worms and viruses.

These aggregate anomalies are important events that need to be detected and corrected, or the system may later be halted and require administrative action, e.g., consider the consequences of a DDoS attack launched via PlanetLab. Unfortunately, while monitoring individual nodes is straightforward, monitoring the aggregate activity of a distributed system can be expensive. This is because all of the individual nodes must exchange information about their state with other nodes to compute the aggregate; the most active tasks at any node will not necessarily correspond to the most active tasks across the system. One straightforward possibility is to periodically query each node and have it send a summary of its activity to a central location, where the aggregate is then computed. These periodic operations require sending n such summaries for a system of size n. Tolerable overhead thus requires a trade-off against collection frequency, so that this method may be slow to detect anomalies instead of expensive. Moreover, the communication cost is incurred whether or not there is anomalous activity, which is wasteful in the common case of normal behavior.

In this paper, we study the problem of rapid and efficient detection of anomalously large aggregates in a distributed system. Our work is geared to produce new mechanisms that can operate to check large aggregates in demanding settings where periodic queries are not sufficient. Such mechanisms would enable new styles of monitoring for distributed systems. Very efficient mechanisms would allow detailed monitoring that covers many kinds of flow aggregates at the same time and at different timescales (e.g., ports and types of TCP packets, IP addresses and /24 prefixes). Very rapid detection would allow systems to provide more open access, since problems can be halted with a "runtime exception" before they cause substantial damage.

We began to tackle this problem by studying the literature, since the general problem of finding frequent items

in streams is well-studied. To our surprise, we found few results that were well-suited to our specific problem. Recent work such as PIER [14] and Sophia [34] also considers the monitoring of large-scale distributed systems. However, at the core of these systems is an engine that distributes queries to all nodes and aggregates the results. This is well-suited to scenarios where the goal is to obtain a snapshot (*e.g.*, "how many nodes saw a TCP connection at port 1434 in the last minute?") or continually record system state (*e.g.*, to study traffic patterns). But in terms of detecting anomalies it is analogous to periodic queries, which we have argued are ill-suited to our task. The most relevant other work is distributed "top-k" monitoring [3], which comes from the database community. This work increases the efficiency with which the k largest aggregates may be tracked. However, from our point of view it is potentially wasteful since it tracks the top aggregates whether or not they are anomalous. Its efficiency further depends on traffic patterns, which can be manipulated by an adversary.

Our new approach makes the system wide collection of data needed to compute aggregates proportional to the level of system activity. As a result, there is little overhead in the common case that there are no anomalies. When there is an anomaly, it is likely to be detected, with larger anomalies being detected more rapidly. We put forth the idea of proportional triggers in a workshop paper [15], suggested possible design strategies and discussed other design considerations. Here, we develop one of those strategies in detail to design the "pull-based" *CoCop(global)* and *CoCop(scoped)*, and make an initial stab at the other strategy to design the "push-based" *CoCop(push)* - all three of which are systems that detect anomalous aggregates. We further evaluate CoCop by analysis and with a trace-driven simulation based on six days of fine-grained PlanetLab flow data. We find it can detect anomalies as quickly as the straightforward approach of periodic queries but with communication costs that are orders of magnitude lower. Although the different CoCop variants are probabilistic/approximate in nature, they are highly accurate on our trace data and do not miss significant anomalies, even if they are widely distributed. Moreover, the simple approach has the virtue of being robust to changes in the makeup of system traffic.

The rest of this paper is organized as follows. We formally define the problem we are tackling in Section 2, and outline our approach in Section 3. We present the design of our pull-based systems in Section 4. We then evaluate these pull-based systems to better understand their properties, using analysis (Section 5) and trace-driven simulation (Section 6). In Section 7, we explore push-based mechanisms and present the design, analysis and simulation results of our push-based system. We discuss the major points to take away in Section 8, contrast our approach with related work in Section 9, and conclude in Section 10.

2 Problem and Goals

In this section, we formally define the problem we study and goals for the design of our solution.

2.1 Problem Definition

We use a motivating problem instance for ease of exposition: the hypothetical use of PlanetLab to launch DDOS attacks. PlanetLab represents a large pool of widely accessible and well-placed nodes that could collectively be used to attack a target by simply having each node send traffic towards the target. This could occur either intentionally or due to an experiment gone awry. If 1000 nodes send at the low rate of 10Kbps then home users would be overwhelmed; if the nodes send at commonly observed rates of 1Mbps then even a highly provisioned target may be overwhelmed. Any of these situations would be highly detrimental, and we want to be able to detect them rapidly, should they occur, so that the traffic can be halted. Note that PlanetLab currently imposes rate-limits at individual nodes on the amount of traffic that can be sent to any destination to mitigate bandwidth overuse and abuse. But this mechanism will not catch aggregate problems, and there is no available mechanism that will.

This scenario captures the core design problem, though we note that the problem we study is more generally applicable, e.g., to incoming traffic and events other than traffic. More formally, let n be the number of nodes in the system, and let a *flow* be a collection of packets across all nodes in the system that share some common characteristic. This may range from IP addresses and TCP/UDP ports to more subtle properties such as the links along the path the packets take through the network. In our PlanetLab scenario a flow is defined by the destination IP address. Let $b_i^x(t)$ be a function of time that gives the bytes sent to flow x (which has destination ip_x in our example) by node i at time t. That is, $b_i^x(t)$ is the contribution of node i to flow x, having non-zero values only at the times at which the node sends a packet that belongs to the flow. Finally, let the threshold at which a flow becomes anomalous be N bytes in T seconds. Intuitively this is a threshold to catch a flow that sends at a rate greater than N/T, but we express this using an interval so that the rate is well defined.

Our goal is to find system-wide flows that are anomalous aggregates because they have sent more quickly than the

threshold. That is, our problem is to solve the following "sliding window" test:

$$\exists\, x, y\ s.t.\ \sum_{i=1}^{n} \sum_{t=y-T}^{t=y} b_i^x(t) > N \qquad (1)$$

A solution to this equation means that flow x is an anomalous aggregate at time y. Note that other temporal weightings may also be natural in some settings, *e.g.*, exponential decay [24]. We do not explore this further in this paper.

2.2 Solution Goals

An ideal distributed detector would possess several properties. These serve as design goals for our solution: efficiency, scalability, timeliness, accuracy and robustness.

The primary goal for our design is that it be efficient. This can have different meanings, but our emphasis is on communication costs rather than computation and storage at a node, which we assume are adequate to the task. We would like the communication costs to be low in the common case that there is no anomaly.

We also seek a design that scales, in the sense of preserving efficiency as the system grows, both in terms of the number of nodes and flows that are being monitored. At a minimum, our solution must be efficient at moderate scales of hundreds of nodes, such as the current PlanetLab deployment. Ideally, beyond this scale, we want the communication costs to increase proportional to the activity of the system, since this seems a plausible lower bound on the required work.

Our timeliness goal is that our design quickly detect aggregate anomalies, despite tradeoffs made to attain the goals of efficiency and scalability. Moreover, it should be accurate such that it is not possible for persistent aggregate anomalies to escape detection.

Finally, we want our design to be robust in the sense that its accuracy in the case that there are anomalies cannot be undermined by controlling the pattern of traffic. This is because we may be dealing with adversaries who will seek to evade detection. Similarly, we do not want adversaries to be able to greatly affect the efficiency of the design in the case that there are no anomalies.

3 Approach

To check if an aggregate is anomalous, the data needed to calculate the aggregate must come together at the same place at the same time. In a *pull*-based mechanism, a node requests ("pulls") readings from all the participants at once. Whereas in a *push*-based mechanism, each node sends("pushes") data toward the accumulating node autonomously. In this paper, we have designed, analyzed and evaluated both kinds of mechanisms, albeit the former in more detail than the latter. Sections 4-6 present the design and evaluation using analysis and trace-driven simulations of our pull-based mechanisms, while Section 7 does the same for a push-based mechanism. However, both kind of mechanisms are based on what we call the "proportional approach", where the system-wide collection of data needed to compute the aggregate of a flow is proportional to the level of aggregate activity of the flow. In the pull-based mechanism, this approach is realized using probabilistic triggers which we now discuss in more detail. The push-based mechanism uses deterministic triggers which we elaborate in Section 7.

3.1 Probabilistic Triggers

In contrast to our approach of collecting data proportional to the level of aggregate activity of the flow, periodic queries collect system-wide data whether or not the aggregate is likely to be large. There is still an apparent "catch-22" in our approach, however, since we do not know the level of aggregate activity before collecting system-wide data, so it would seem difficult to make collection proportional to this activity. This is not so. Each node can act independently to trigger a system-wide collection with a probability that depends on its local activity. The overall behavior will then be the sum of the actions of individual nodes, which will be related to the system-wide activity in an expected sense. By suitably adjusting the probabilities (as given in the next section) we can arrange for computation of the aggregate to be expected to occur if there is an anomaly.

In the sections that follow, we will show that simple, independent triggers have three desirable properties. First, not only is computation of the aggregate expected to occur if it is anomalous, but it is increasingly less likely to occur if the aggregate is increasingly small. By definition, most flows will be significantly smaller than the anomaly threshold, and in real systems there are typically many small flows (as we see in PlanetLab traces). This means that collection is unlikely for most flows, which lends our approach efficiency. It also implies scalability, since the communication cost of collection is tied to system activity.

Second, larger aggregate anomalies are more likely to be detected, since the chance of a collection increases with the value of the aggregate. It is precisely these significant anomalies that we most wish to detect; an aggregate that is anomalous by definition because it is just over the threshold is little different in practice than an aggre-

gate that is not anomalous because it is just under the threshold. Moreover, this probabilistic form of detection seems suited to our problem, whereas in other settings the lack of a guarantee may be unacceptable. The chance of detection is independent for each period that is checked for anomalies. Thus, a persistent anomaly will soon be caught, and as before it is precisely these anomalies that we most wish to detect; an aggregate that briefly exceed the threshold then ceases to do so does not require corrective action. To capture these factors, we define the *excess* of an anomalous aggregate to be the traffic it sends before its detection that is beyond the permitted threshold. To be accurate, our design aims to minimize excess rather than the time to detection. This weights the time to detect larger anomalies more heavily than smaller ones.

Finally, the collection behavior depends on the overall level of activity of a flow, rather than its distribution across individual nodes. This means that widely distributed anomalies are as likely to be caught as concentrated ones. It contributes to robustness since an adversary cannot manipulate traffic in the system to lower their chance of detection.

4 Design of Pull-based CoCop

In this section we sketch the design of pull-based CoCop. We begin with the core algorithm and then describe the system that is built around it.

4.1 Algorithm

We define two variants of the algorithm, Global and Scoped, that we study. In both, nodes independently run trigger logic to decide whether to compute and check the aggregate. They differ in how they query to check the aggregate. We describe the trigger logic and then each variant in turn.

4.1.1 Trigger Logic

Recall that our problem definition seeks to determine when a system-wide flow x sends more than N bytes in T seconds. For convenience, let $C = N/T$ be the anomaly threshold rate, which is well-defined for periods greater than T. Let our *trigger interval*, τ seconds, be a multiple of T seconds.

In our trigger logic, each node monitors its outgoing traffic. Periodically, at time t that is a multiple of the trigger interval of τ seconds, each node i computes the average rate of traffic f it has sent to each flow x in the past τ seconds:

$$f_i^x(t) = \frac{1}{\tau} \cdot \sum_{t'=t-\tau}^{t'=t} b_i^x(t') \tag{2}$$

Each node then compares its contribution to the flow, f_i^x (we drop t for brevity), with the system-wide threshold C. The node uses the fraction of contribution as a probability to determine whether it will trigger:

$$p_i^x = \begin{cases} 0 & \text{if } f_i^x < C \cdot \epsilon \\ \frac{f_i^x}{C} & \text{if } C \cdot \epsilon \leq f_i^x \leq C \\ 1 & \text{if } f_i^x > C \end{cases} \tag{3}$$

The lower cut-off at $(C \cdot \epsilon)$ is motivated by the observation that if each node is contributing less than C/n then the aggregate across n nodes must be less than C. That is, we preferentially avoid triggering given extremely small local contributions. The use of ϵ instead of $1/n$ is to control the transition between these regions. While this cut-off is low, we will nevertheless see that it is valuable in Section 6 when dealing with real measurements because of the highly skewed nature of flows, in particular with very many tiny flows.

4.1.2 Global Queries

In CoCop(global), once a node triggers for flow x, it issues a global query to ask all the other nodes, j, for their local rates f_j^x for the suspect flow. With the responses, the triggering node can trivially compute the aggregate for flow x, F_x, as the sum of all the individual contributions to destination x, f_i^x. It then checks whether the aggregate exceeds C. If so, the node alarms and begins to apply its enforcement function (which is a system policy, discussed in the next section).

With this algorithm, if even a single node triggers, then the global query will be run and the aggregate will be computed. The calculation of the trigger is such that one trigger is expected if the aggregate is anomalous. There is also the issue of suppression of multiple simultaneous triggers. These should be combined into a single global query operation. To do so in a fully distributed manner, we borrow from SRM [10]: nodes wait a short randomized interval between the time they trigger and the time they issue a global query, and they defer if they hear another node has issued a query for their flow. In this manner it is likely that only one global query will be issued.

4.1.3 Scoped Queries

The expensive operation in CoCop(global) is the global query. CoCop(scoped) aims to reduce the number of global queries based on the intuition that queries across smaller numbers of nodes can be used to gain confidence in whether there is likely a system-wide problem. Of

course, a node can never be sure that is no anomaly without computing the overall aggregate, so we are trading reduced communication overhead for an increased risk of missing an anomaly.

In CoCop(scoped), a triggering node asks a random subset of the system for their contributions. The size of this subset is related to its fraction of the aggregate threshold. If the sum of these contributions is greater than their fair share of C (with each node entitled to C/n) then this process repeats and asks an ever larger subset of the system. This continues until either the global aggregate has been checked and found to be anomalous, or the scoped query terminates early when it falls below its fair share. To keep the number of iterations low, we round the size of the subset up to the nearest power of two. In this manner, there can be no more than $\log n$ steps for n nodes.

More formally, a node i that triggers starts with a *view* $V_0 = \{i\}$ of only its contribution which observes the global aggregate as $G_i^{V_0} = \sum_{j \in V_0} f_j$. This observation of the global aggregate is above its fair share if $G_i^{V_0} > \frac{|V_0| \cdot C}{n}$. In this case, the node extends its view to $2^{\lceil \log \frac{n \cdot G_i^{V_0}}{C} \rceil}$ nodes, a set large enough that it may be below the fair share in aggregate. We refer to this view as V_1. Once the node hears from the nodes in V_1, it calculates $G_i^{V_1} = \sum_{j \in V_1} f_j$. Again, it checks to see if this is over the fair share, *i.e.*, if $G_i^{V_1} > \frac{|V_1| \cdot C}{n}$. If so, the node extends its view further to $2^{\lceil \log \frac{n \cdot G_i^{V_1}}{C} \rceil}$ nodes that constitute V_2. This process continues until either the aggregate check completes or is halted.

Observe that in the case of a detected anomaly this process makes a global query that is spread over time. However, small aggregate values are increasingly likely to terminate early, saving the work involved in a global query. Also, note that the suppression of multiple simultaneous triggers is not included in CoCop(scoped) because each node grows its own set of nodes. Instead, it should be possible to merge global queries that collide, if they prove problematic. Finally, this variant implicitly contains an ϵ-cutoff of $1/n$, since the scoped query will not proceed unless the node is greater than its fair share.

4.2 System Components

Figure 1 shows the architecture of pull-based CoCop running on each node. There are four main components:

- *Monitor*: All packets leaving the node pass through the monitor. It logs the bytes sent to each flow.
- *Summarizer*: The summarizer reads the logs produced by the monitor, produces an activity summary, and runs the trigger logic described in Section 4.1 to decide whether to issue a query for the aggregate. It runs every trigger interval of τ seconds.
- *Collector*: This component handles inter-node communication. It acts as both a server and client, spawning an aggregation query when requested by the summarizer. As part of this query it sends requests to other nodes for a particular flow, collects the responses, and uses them to determine if the aggregate is over the threshold.
- *Enforcer*: This component is notified when an anomalous aggregate is found by the collector. The action it takes is a system policy. We describe some possibilities shortly.

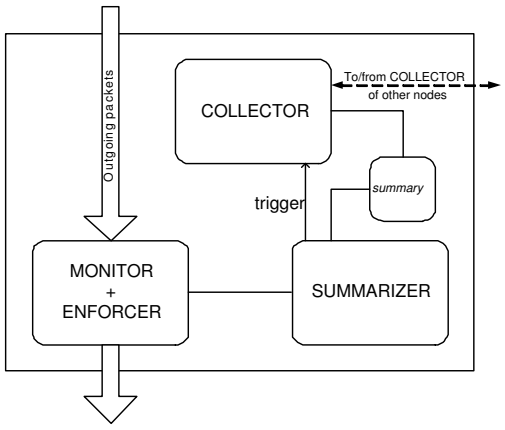

Figure 1: Components of pull-based CoCop at each node

4.3 System Communication

CoCop needs two communication mechanisms to implement the components described above. Both mechanisms solve well-known problems, and so we do not innovate but instead borrow existing solutions.

First, the collector must send a request to all other nodes in the system. Simple one-to-one strategies scale up to on the order of hundreds of nodes (and were workable in our PlanetLab experiments). Beyond this, distribution trees with better scaling properties are needed, such as any multicast scheme. Additionally, in-network aggregation is needed in the reverse direction to combine responses from individual nodes.

Second, nodes in pull-based CoCop require knowledge of the other nodes that are participating in the system. That is, the nodes should run a membership protocol. This information is used in two ways: to send queries to other nodes; and for knowledge of the system size to estimate the fair share (of C/n). We do not expect this to particularly demanding because approximate membership should be sufficient.

4.4 Enforcement Policies

Our focus is on detecting anomalous aggregates, but once an anomaly is found then something must be done. One possibility is to halt the anomalous flow until an administrator allows it to continue. This may be reasonable when anomalies are clearly dangerous, e.g., a malicious attack. But it is likely to be too severe a response when the anomaly is a temporary excess of resource usage. In this case, we would like to cap the aggregate resource usage to provide a result closer to a distributed form of rate-limiting.

To do so, we may limit each node to the fraction of its current usage that sums across all nodes to the threshold minus a penalty. This is slightly more complicated than limiting each node to its fair share, but better handles nodes with uneven contributions (as long as the level of contribution of a node does not change greatly).

A final consideration is the period for which the limit is enforced. As before, we would rather use an automated mechanism than have the system frozen until an administrator intervenes. To do so, we can use exponential backoff. That is, we double the punishment period for an anomalous flow with each subsequent violation that is not preceded by a sufficiently long period of good behavior. This scheme has the desirable property that even persistent anomalies cannot send a large amount of traffic above the threshold.

5 Analysis of Pull-based CoCop

In this section we theoretically analyze pull-based CoCop. We begin by formally defining *communication overhead* and *average excess traffic* as our metrics of concern. We then analyze our algorithms in terms of these metrics. In particular, we find the expected communication overhead over a trigger interval, and the worst-case excess traffic for an anomalous aggregate. Later in Section 6, we will corroborate our bounds with simulation results.

The two metrics that we use to assess our algorithms are:

Communication Overhead: This is the wide-area communication (in bits per second) that is exchanged between the participating nodes to find the aggregate flow values, send alarms upon detecting anomalies, etc.

Average Excess Traffic: Excess Traffic is the volume of traffic (in bytes) beyond that allowed by the threshold that the anomaly sends before it is detected. Thus, excess traffic for an anomaly captures the "cost" of not having caught it in a timely manner. Average excess traffic is the average of excess traffic over all anomalies. Let t be the time required to detect an anomalous aggregate flow transmitting at a rate above the set threshold, C (in bits per second); and let F^x be the average rate (in bits per second) of that anomaly over the period until it is caught. Then, its excess traffic is $(F^x - C) \cdot t$.

5.1 Analysis - CoCop(global)

To simplify the analysis, ignore the ϵ-cutoff, *i.e.*, in equation (3), assume $p_i^x = \frac{f_i^x}{C}$ even when $f_i^x < C \cdot \epsilon$.

Consider a flow x and let the aggregate traffic on it, $\sum_i f_i^x$, be F^x. At the end of a trigger interval, in an expected sense, $\sum_i p_i^x = \sum_i f_i^x / C = F^x / C$ nodes will initiate a global query for flow x.

Thus, in the common case that the aggregate traffic for a flow is much below the threshold C, it is unlikely that a global query will be issued for that flow. However, when there is an anomaly, *i.e.*, $F^x > C$, atleast one node is expected to trigger a global query. Multiple queries are avoided by *suppression*. Each aggregation requires n to-and-fro messages. Thus, assuming that suppression works perfectly, the total expected communication overhead of the system is $2 \cdot n \cdot \sum_x max(F^x/C, 1) \cdot \frac{64}{\tau}$. The factor of $64/\tau$ is to account for size of query request and reply being 8 bytes and to convert units from messages per trigger interval τ, to bps. We then have:

$$\text{(overhead)} \quad = \quad 2n \cdot max(F/C, \#(flows)) \cdot \frac{64}{\tau} \quad (4)$$

This is proportional to the total activity level in the system, $F(= \sum_x F^x)$. Only in the worst case that all flows in the system are anomalous does this overhead approach that of global querying (with the same period as the trigger interval).

To calculate the excess traffic, consider an anomalous flow x. Observe that since nodes are independent, the probability that no node triggers aggregation at the end of a trigger interval is $\prod_i (1 - p_i^x)$; and similarly until the end of $(k-1)$ intervals, is $\prod_i (1 - p_i^x)^{(k-1)}$. The probability, $P_{k\tau}$ that some node triggers for the first time in the k^{th} trigger interval is the probability no node has triggered a query until $(k-1)^{th}$ multiplied by the probability that atleast one node does so in the k^{th}, *i.e.*

$$P_{k\tau} = [\prod_i (1 - p_i^x)^{(k-1)}] \cdot [1 - \prod_i (1 - p_i^x)]$$

This gives the expected time after which some node triggers a global query:

$$\sum_{k=1}^{\infty} k\tau \cdot P_{k\tau} = \tau / (1 - \prod_i (1 - p_i^x)) \quad (5)$$

Now, since x is an anomaly, $\sum_i p_i^x (= F^x/C) \geq 1$. Given this, $\prod_i (1 - p_i^x)$ takes its maximum value when $\forall i, p^x = 1/n$; and $(1 - 1/n)^n$ can be shown to have an upper bound of e^{-1}. Thus, even in the worst case, which corresponds to the situation when all nodes send just above their fair share, the expected excess traffic, which is $(1/8) \cdot (F^x - C) \cdot \sum_{k=1}^{\infty} k\tau \cdot P_{k\tau}$ is bounded from above by:

$$\text{(excess traffic)} \quad = \quad \frac{(F^x - C)\tau}{1 - e^{-1}} \cdot \frac{1}{8} \qquad (6)$$

The factor of 8 in the denominator is to convert units from bits to bytes. Note that the worst case bound depends only on the F^x and not on individual f_i^x's. Thus an adversary can not fool the system by spatially varying the nodes which contribute to the aggregate. Also, the entire triggering logic is based only on observations over the last epoch, which makes it less amenable to attacks that exploit temporal variations.

Qualitatively, the ϵ-cutoff significantly cuts down the communication cost because of skew towards very small flows in real systems, but increases excess traffic because of potentially longer time-to-detection. Since, the overheads only reduce by introducing the ϵ-cutoff, the actual overhead of CoCop(global) is bounded by equation (4). Let us study the effect of excess traffic more closely. Out of F^x, let F'^x be the contribution of nodes which are above ϵ-cutoff. Then, the time to detection for traffic F^x with ϵ-cutoff is no more than the time-to-detection for traffic F'^x without ϵ-cutoff. The latter, call it T', can be calculated using equation (5). The excess with cutoff is then bounded by $\frac{1}{8}(F^x - C) \cdot T'$.

5.2 Analysis - CoCop(scoped)

As before, consider a flow x and let the aggregate traffic sent on it, $\sum_i f_i^x$, be F^x. At the end of each trigger interval, in an expected sense, $\sum_i p_i^x = \sum_i f_i^x/C = F^x/C$ nodes will initiate a scoped query for flow x. The node initiating a scoped query will in the worst case (for overhead) ask all the non-zero contributors to the aggregate and thereby end up asking $2^{\log\lceil n \cdot F^x/C \rceil}$ of the nodes. The expected overhead for a particular flow is the product of the probability that a node will initiate a scoped query and the number of nodes that may be asked for the scoped query of that flow; and therefore is bounded by $2 \cdot (F^x/C) \cdot 2^{\log\lceil n \cdot F^x/C \rceil} \cdot \frac{64}{\tau}$. Hence, the total overhead of the system is bounded by:

$$\begin{aligned} \text{(overhead)} \quad &\leq \quad 2 \cdot \frac{64}{\tau} \cdot \sum_x (F^x/C) \cdot 2^{\log\lceil n \cdot F^x/C \rceil} \\ &\leq \quad 4n \cdot \frac{64}{\tau} \cdot \sum_x (F^x/C)^2 \qquad (7) \end{aligned}$$

where the expression on the right-hand side is a weak upper bound of the left-hand side. Note that in the common case that $F^x < C$, the overhead is even lower than that of CoCop(global). The overhead is more when the aggregate is above the threshold, but since that, by definition, happens very infrequently, the average overhead remains much lower.

To calculate the excess traffic, let us again consider an anomalous flow x. In CoCop(global), if a node initiates a query, it learns whether there is an anomaly or not since it asks all the other nodes for their contributions. However, in CoCop(scoped), the initiator might choose nodes which are not sending any traffic though there might still be other other nodes which are sending a large amount of traffic. Given that node i initiates a scoped query for flow x, let q_i^x be the probability that a node ends up asking all other nodes (and therefore detects the anomaly). Then, using an argument similar to the above, the expected excess traffic is:

$$\text{(excess traffic)} \quad \leq \quad \frac{(F^x - C)\tau}{1 - \prod_i (1 - p_i^x \cdot q_i^x)} \cdot \frac{1}{8} \qquad (8)$$

However, calculating a precise closed-form expression for q_i^x turns out to be difficult. Using Chebyshev's bounds to bound the probability, still does not give a closed-form expression (and Chernoff bounds give a very loose bound in this setting). So, in order to analyze this expression, we consider a very simplified model where $(F^x/C)k$ nodes are at C/k each and the rest are at zero. This gives us a lower bound of $1/k$ for q_i^x (see appendix); and hence an upper bound of $\frac{1}{8} \cdot (F^x - C)\tau/(1 - e^{-1/k})$ for the excess traffic.

Since $(F^x/C)k \leq n$, therefore $k < n$ and so, for a given n, this expression can be shown to be bounded from above. Thus under the simplified model, the excess traffic for for CoCop(scoped) is bound - though this bound is much weaker than the tight bound we gave for CoCop(global).

As in CoCop(global), the logic depends only on the observations over the last epoch. Therefore, it too is resistant to temporal exploits. It is more difficult to argue that it is resistant to spatial variations without a generic closed form expression; but intuitively, since the set of nodes asked at a step is chosen randomly, that should be the case.

Introducing the ϵ-cutoff has does not change the results because if the local contribution of flow x at node i, f_i^x, is less than the ϵ-cutoff (and hence less than its fair-share of the global threshold, because $C \cdot \epsilon < C/n$), then n does not issue a query to any other node; and this is exactly what happens by introducing the ϵ-cutoff too.

6 Simulation Results

In this section, we use traces of PlanetLab flow data to evaluate the efficiency, timeliness and accuracy of pull-based CoCop.

6.1 Data and Setup

We collected traces over a period of 6 days in January 2005 from 337 PlanetLab [1] nodes. These traces record, at a 1-second granularity, the volume of traffic (in packets and bytes) that each node sends to each destination address. The traces are generated using a module built on top of ulogd [2] which allows userspace packet-logging via the ULOG target in iptables/netfiler.

To use these traces to test our algorithms, we must define what we mean by flow and specify the anomaly thresholds. We use destination IP address to define a flow. We study the traces and pick a threshold (80 Mbps) that treats a small fraction of the highest rate aggregates as anomalous; a later subsection considers the effect of varying this threshold. This seems a more demanding test than adding synthetic anomalies to our traces. We further pick an ϵ-cutoff of 0.1% of the threshold. This is roughly a third of the node fair share given the size of the system.

6.2 Characterization of Aggregates

We begin by characterizing the aggregates in our trace data along several dimensions. We observe how aggregates vary across flows, and how they are distributed across nodes and across time.

6.2.1 Variation Across Flows

In real networks, flows tend to be skewed, and this skew impacts the effectiveness of different designs, e.g., caching. To test for this skew, we study the distribution of the magnitude of aggregate flows at different points in time (Figure 2). For each second traced in the dataset, we calculate the total traffic for each flow from all the nodes combined. There were nearly 2.1 million distinct IP destinations to which one or more PlanetLab nodes sent traffic during our trace, which resulted in 1067 million one-second aggregate flow samples. The graph plots the distribution of these samples on a log-log scale.

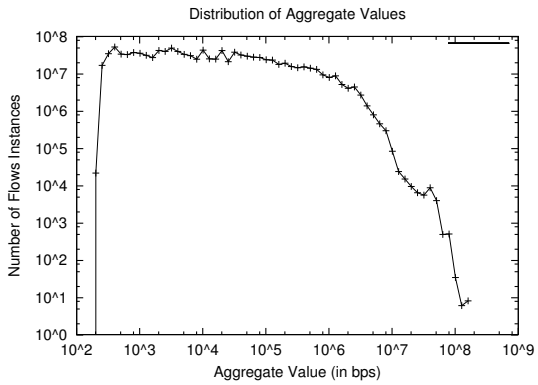

Figure 2: Distribution of Aggregate samples. The distribution is skewed, with very many small samples and very few large samples. Note the log-log scale.

Threshold	#Aggregates	#Samples
$>$ 0 Mbps (Total)	2.1M	1067M
$>$ 40 Mbps	135	13736
$>$ 60 Mbps	66	1176
$>$ 80 Mbps	42	547
$>$ 100 Mbps	2	50

Table 1: Number of aggregates and aggregate samples above different thresholds

We see that there are a very large number of flow samples for which only a small amount of aggregate traffic is seen; this number falls quickly as the aggregate traffic increases but spans more than five orders of magnitude. That is, the distribution is highly skewed and there are both very large and very small aggregates. To examine the tail more closely, Table 1 lists the number of distinct aggregate flows and the number of aggregate flow samples for different thresholds in the tail of the distribution. From this tail we somewhat arbitrarily pick 80 Mbps as a threshold, as it leaves a sufficient number of larger aggregate samples for our experiments. We study the effect of varying the threshold later in this section.

6.2.2 Spatial Spread of Aggregates

A property of specific interest to us is the spread of aggregates across nodes, since our system finds widely distributed aggregates. To see this spatial spread, we take the anomalous flows (the 42 flows that exceeded our threshold of 80 Mbps during the trace) and plot them as CDFs in Figure 3. Each CDF shows the fraction that each PlanetLab node contributes to an aggregate, ordered with the

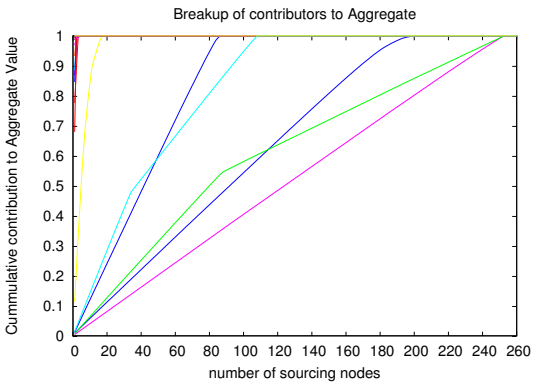

Figure 3: Cumulative composition of anomalous flow aggregates by PlanetLab node. Most anomalies have only one significant contributor and the curves for these overlap as a step function at n=1. The slanted curves are distributed anomalies.

largest contribution first for each aggregate. Thus a vertical line over the y-axis is an anomaly made up of a single node, while a 45^o line is an anomaly spread evenly across all nodes.

We observed a range of scenarios. In the majority of cases, there are only one or two contributors. Some aggregates were comprised of tens of contributors, and a few were very widely distributed across hundreds of nodes. Further, although the total aggregate for these widely distributed flows was well above the threshold (some exceed 100 Mbps), the highest value that an individual node saw was low in many cases with one as little as 470 Kbps. Thus, the traces contain flows that appear benign when viewed from the perspective of individual nodes but are anomalous in aggregate.

6.2.3 Temporal Spread of Aggregates

We are also interested in how the anomalies vary over time, since relatively steady aggregates lend themselves to history-based methods. To explore this, we plotted the variation in the values of anomalous aggregate flows. All of the aggregates show significant variation over the period. Figure 4 shows the plots of four of the 42 anomalies. We picked these plots to show a sample of behavior as we could not discern a common trend: some were transient while others persisted over a long periods of time; some anomalies greatly exceeded the threshold, others were very close to it.

This temporal variance has two implications. First, the existence of persistent anomalous aggregates suggest that there indeed is a need for mechanisms that quickly detect (and respond) to them. Second, making predictions about future behavior of flows based on their history seems difficult, which again suggests that a monitoring mechanism is useful.

6.3 Performance of Pull-based CoCop

We now use the PlanetLab traces to evaluate the efficiency and timeliness with which pull-based CoCop can detect anomalous aggregates. We use the metrics of communication overhead and excess traffic defined in Section 5. In the absence of any other system specifically designed for this task, we compare our system with two practices that lie at opposite extremes in terms of what they try to achieve:

- *Global Pull*: Periodically, a central coordinator asks all the nodes for the volume of traffic that they have sent to each flow in the last period.
- *Heavy-hitter*: Each node runs a heavy-hitter algorithm [13, 9](anyone will do) and sends information to the coordinator only for the top flows it sees locally instead of all the local flows.

Global Pull ensures that if there is an aggregate that was anomalous during an interval, then it will be detected at the end of that interval. However, it incurs a large communication overhead as a result of sending information for all flows. Thus it trades overhead for bounded excess traffic. Heavy-hitter algorithms reduce communication costs significantly by only sending information about the largest local flows. However, anomalous aggregates with many contributors may then go undetected for an indefinite period.

The choice of the trigger period, τ, plays an important role in this balance. The smaller this time interval, the greater the communication overhead, but the sooner the anomaly will be caught. We study overhead and excess first, then consider the effect of varying the trigger period. Note that to better evaluate our detectors we run our system without enforcement by forgetting an anomaly as soon as it is found and searching it all over again. Enforcement would simply reduce the effective size of our dataset.

6.3.1 Communication Overhead

We begin by studying the communication overhead of the system. Figure 5 compares the schemes by plotting the log of communication overhead (in Kbps) for different trigger intervals. The overhead for all mechanisms falls as the trigger interval increases from one second to ten

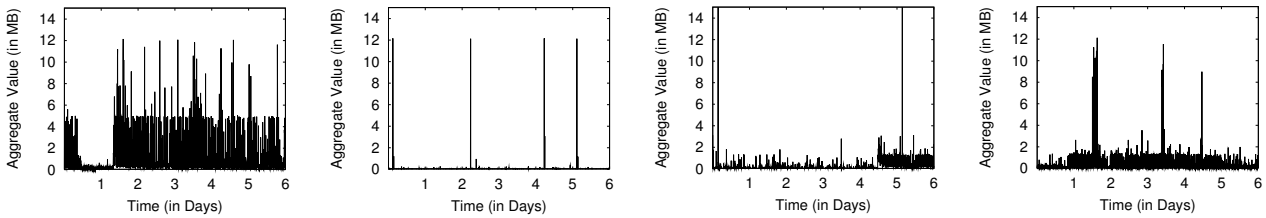

Figure 4: These figures show the temporal variation of four of the 42 aggregate flows that were anomalous at some point during the trace. In general, no common temporal characteristic is observed amongst the anomalous flows.

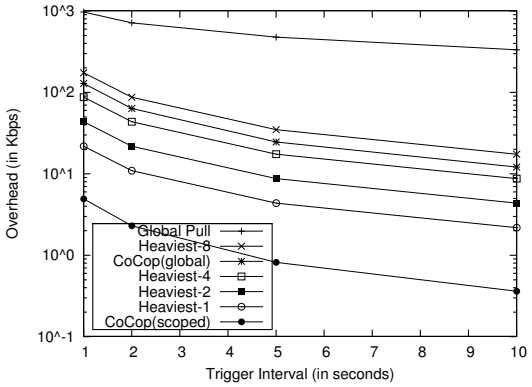

Figure 5: Communication Overhead. The overhead of CoCop(global) is approximately the same as heaviest-5 and about 10 times less than periodic pull. CoCop(scoped) is almost an order of magnitude less than reporting the heaviest flow.

seconds and beyond. The fall is close to inversely proportional but not exactly so. This is because as trigger interval falls from ten to one, the number of flows does not decrease by a factor of 10, but falls somewhat more slowly.

Among the different mechanisms, global pull incurs the most overhead. The overhead of the heavy-hitter algorithms depends on how many *heavy-k* flows the nodes report, but even for $k = 4$, the overhead is nearly an order of magnitude lower than periodic global pull. CoCop(global) incurs about as much overhead as reporting the heaviest 5-6 flows. Finally, CoCop(scoped) is almost another order of magnitude more efficient than reporting just the heaviest flow. Intuitively, this is because, CoCop(scoped) does not necessarily initiate a query even for the local heaviest flow, unless it is suspect.

The above results incorporate the ϵ-cutoff, where nodes do not trigger for very low flow values. To study its effect, we also ran CoCop(global) the cutoff. We found that removing the cut-off increased the overhead by almost 25%. This is because of the skew in the distribution of aggregate values (Figure 2) which included many extremely small flows.

Finally, we found the experimental overhead to be in good agreement with our analysis. The average F value of the aggregates of all concurrent flows over the trace was about 288 Mbps. This translates to 31 Kbps using equation (4). The actual overhead for CoCop(global) is somewhat lower at 24 Kbps. However, the difference between the two is very close to the 25% we observed from the ϵ-cutoff, which the analysis of communication overhead ignores.

Similarly, plugging in values from the traces in equation (7) gives an overhead of 0.92 Kbps which indeed bounds the experimental overhead of 0.8 Kbps for CoCop(scoped).

6.3.2 Accuracy

Due to their probabilistic nature, our algorithms are not guaranteed to detect all the anomalous aggregates that periodic querying detect. We define the *relative accuracy* of a mechanism to be that fraction of anomalies detected by periodic querying, which is eventually detected by the mechanism as well. Table 2 shows the relative accuracy of different mechanisms with varying trigger intervals.

Trigger Interval	1s	2s	5s	10s
CoCop(global)	1.0000	0.9866	1.0000	0.9778
CoCop(scoped)	0.9961	0.9859	0.9744	0.9778
Heaviest-1	0.9940	0.9860	0.9744	0.9767
Heaviest-2	0.9963	0.9913	0.9881	1.0000
Heaviest-4	0.9982	1.0000	1.0000	1.0000

Table 2: Accuracy (relative to periodic querying) of anomalies detected by different mechanisms

Both variants of CoCop are quite accurate. In all experiments they detected more than 97% of the anomalies that periodic querying did. Both were more accurate than reporting only the heaviest local flow. We were surprised, however, that heaviest-2 and heaviest-4 had such high relative accuracy, since they may miss widely distributed

aggregates. After investigating our data, we suspect that they do well because the system is lightly loaded most of the time. As the load grows, they are likely to resemble heaviest-1.

Whereas anomalous aggregates are defined in terms of a "sliding-window" test (see equation (1)), all the mechanisms discussed till now use "hopping-window" semantics, *i.e.*, nodes do local computation at the end of each successive, non-overlapping trigger interval. As a result, although the accuracy of CoCop and heaviest-k is bounded above by that of periodic querying, there are some "true" anomalies that are not detected even by periodic querying. For example, consider an aggregate flow sending at rate C which starts 0.2τ seconds after the last trigger interval started and lasts till 0.2τ seconds after that interval ended. This is an anomaly under the sliding-window semantics, but is not detected by periodic querying as the aggregate is less than the threshold C in both the trigger intervals during which the aggregate was sending traffic. We studied the occurences of such anomalies in the traces that were not detected even by periodic querying. Periodic querying showed an accuracy of 0.9957, 0.9883 and 0.8823 for trigger intervals of 2s, 5s, and 10s respectively; where the (absolute) *accuracy* of a mechanism is defined to be the fraction of anomalies that it eventually detects. We could not find the accuracy corresponding to a trigger interval of 1s because the traces were collected at a granularity of 1s and so instead of running the sliding-window test with windows sliding continuously, we could slide them only in discrete steps of 1s each. The decreasing trend in the accuracies is explained by the fact that the larger the trigger interval, the more the likelihood of having a "true" anomaly spread across two consecutive intervals in such a manner that it is anomalous in neither.

6.3.3 Excess Traffic

With significant savings on communication overhead and having reasonable accuracy, one might expect that CoCop will pay a large penalty in term of excess traffic. However, this is not the case. The average excess traffic sent before detection of an anomalous aggregate is plotted in Figure 6 as a function of the trigger interval.

Note that the larger the trigger interval, the longer the minimum time before an anomaly can be detected, and hence the larger the average excess traffic. This trend is common across all mechanisms. Also, the curve corresponding to global pull is a lower bound on the curves for the other mechanisms. This is because, in global pull, the coordinator has an system-wide view of all the flow aggregates at the end of every trigger interval. Thus for a given trigger interval, any anomaly is detected as soon as it possibly could have been.

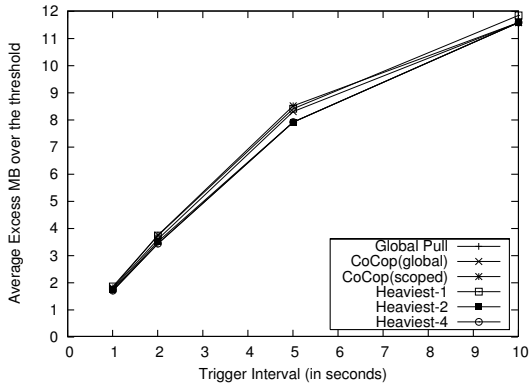

Figure 6: Average Excess Traffic. The lowermost curve is Periodic Pull, which is the best a mechanism can hope to achieve. CoCop and heaviest-k are only very slightly worse, though for different reasons.

The graph shows that the average excess traffic of CoCop (both with global and scoped queries) and the heavyhitter algorithms is only marginally greater than that of global pull. As before, we suspect that the heavyhitter is performing well because the system is lightly loaded.

The reason for CoCop's good performance is different. Because of its proportional approach, anomalies well above the threshold are almost always detected at once. It is typically only aggregates just above the threshold that remain undetected for any period of time. The small increase in the average excess traffic is due to these aggregates.

We test the analytical bounds on excess traffic that we got in Section 5. As compared to periodic querying, simulations showed an increase by factor of 1.049 in the average excess traffic for CoCop(global). This is much lower than the tight analytical worst bound of $1/(1-e^{-1}) \approx 1.58$ (from equation (6)), because there were only a few widely distributed anomalous aggregates in the traces, to which the worst case corresponds to. CoCop(scoped), like CoCop(global), also showed only a slight increase of factor of 1.076.

6.3.4 Tradeoff and Ordering

The preceding results hint at the inherent trade-off between the communication overhead and the excess traffic. As the trigger interval τ increases, the communication overhead falls but the excess traffic increases, and vice versa. Figure 7 highlights this tradeoff by plotting the (log of) communication overhead versus the excess traffic for the different mechanisms.

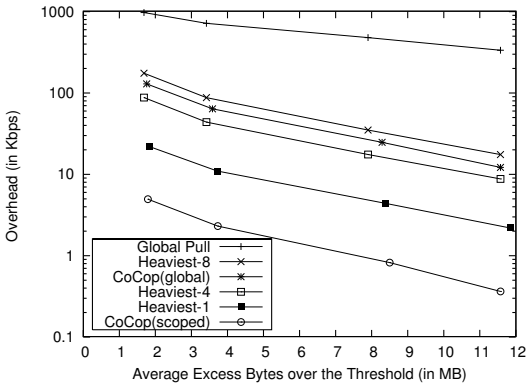

Figure 7: Tradeoff between Average Excess Traffic and Communication overhead for the different mechanisms. The graph illustrates the win-win situation for CoCop(scoped) over the rest.

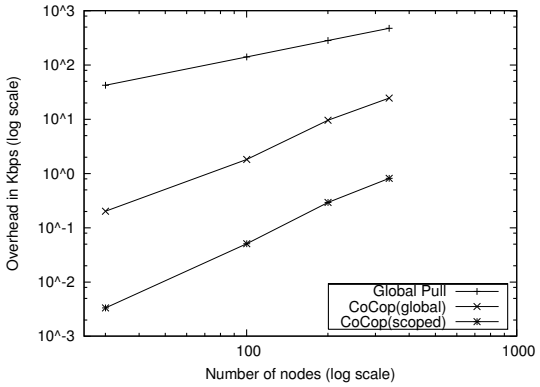

Figure 8: Variation of communication overhead with number on nodes

This graph allows us to compare different schemes in terms of both metrics at once: a point to the left and lower of another point is superior to it since it has both less overhead and a lower excess. The key result of this graph is that there is a strict ordering of the mechanisms we have studied. CoCop(scoped), our best mechanism, is always to the left and lower than any other curve. This implies that it can beat the operating point of any of the above schemes (for our dataset at least, but we presume more generally) by a suitable choice of operating point.

6.3.5 Scalability

Next, we evaluate how pull-based CoCop scales with system size. Figure 8 plots the communication overhead of CoCop as we vary the number of nodes from 30 to 337, keeping the threshold fixed. These simulations were run for a trigger interval of 5 seconds. We also plot the curve for global-pull; any periodic approach (global-pull, heaviest-k) will scale linearly with size of the system.

The graph shows that CoCop always has less overhead than global pull. Under scaling, in the worst case it degrades to global pull. We see that the increase in communication overhead of CoCop is greater than that of periodic querying. This is expected given that we have fixed the threshold, and is in good agreement with our analysis. As we increase the size of the system, its activity will also increase roughly linearly. With a constant threshold, our analysis predicted that the communication overhead of CoCop(global), $(2 \cdot n \cdot F/C)$, would grow roughly quadratically. We observe the same underlying behavior for CoCop(scoped). Conversely, by raising our threshold with the size of the system, we would expect to improve the scaling of CoCop(global). We consider this next.

6.3.6 Variation with Threshold

The performance of CoCop depends on the threshold that is chosen. To study this, we plot the variation of both the average excess traffic (left y-axis) and overhead (right y-axis) as we vary the threshold in Figure 9. The curves for the excess traffic of both CoCop(scoped) and CoCop(global) were very close to each other and we plot CoCop(scoped) only to avoid clutter.

Recall that our analysis predicted overhead that was inversely proportional (linear in CoCop(global) and quadratic in CoCop(scoped)) to the value of the threshold C. This is what we see. It reflects our original design that allowed CoCop to be effective at reducing overhead in exchange for targeting anomalies that constitute only a small fraction of the total number of aggregate flows. As the threshold decreases and a larger fraction of the aggregates are considered anomalous, the overhead of CoCop increases. However, this increase is bounded – in the worst case it will degrade to periodic querying.

Our analysis also predicts average excess traffic to be proportional to the excess rate beyond the threshold, $(F^x - C)$, and inversely proportional to the number of anomalies. Both of these factors are affected by a change in the threshold in opposite ways. The combined effect is to cancel out both factors, and there is no discernible trend in Figure 9.

7 A Push-based Mechanism

In the previous sections we saw the design and evaluation of two *pull*-based mechanisms to detect anomalies. Once a node triggers for a flow, it issues a query requesting ("pulling") readings from all the other nodes at once. An alternative to this kind of synchronous pull is to have

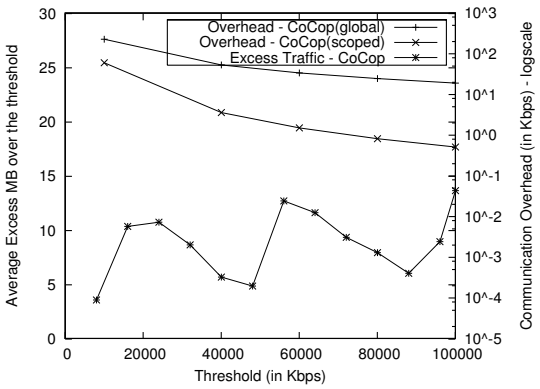

Figure 9: Effect of varying the threshold. Average Excess Bytes in the left y-axis and communication overhead on the right y-axis. While excess traffic shows no discernable trend, communication overhead increases as threshold decreases.

each node autonomously send ("push") data towards an accumulating node, whenever its local trigger fires.

There is a large design space to be explored in the context of push-based mechanisms. There are two major choices to be made in the mechanism - the algorithm running at each node for sending off a trigger, and the algorithm running at the node accumulating these triggers to check if the aggregate is anomalous - for both of which there are numerous possibilities to choose from. In this paper, we take an initial stab at exploring this space by designing and analyzing a push-based mechanism, CoCop(push). We also compare it to the pull-based mechanisms we studied earlier.

7.1 Design

CoCop(push), very briefly, works as follows. Whenever a node sees some fixed amount of traffic on a flow, it sends a trigger to the node coordinating the aggregate-estimation and the anomaly-detection for that flow. If a coordinator node receives a large number of triggers over a short period of time, it infers that there may be an anomalous aggregate and after confirming from all nodes which reported, it raises an alarm.

We now present this algorithm more formally and briefly describe the various system components.

7.1.1 Algorithm

Let the responsibility for maintaining the aggregate value for flow x and detecting if it is anomalous, be on a node that we call the "node coordinating for flow x", $N(x)$.

Each node i maintains a counter b_i^x of the traffic sent to each flow x, which get reset to zero after every τ seconds. But during those τ seconds, whenever the counter of node i for flow x exceeds the node's fair-share traffic-volume for that interval, $C\tau/n$, it sends a trigger to $N(x)$ for flow x and decreases the counter by $C\tau/n$. If a coordinating node, $N(x)$, receives n triggers for flow x in less than the last τ seconds (sliding-window), then that flow aggregate may be anomalous. $N(x)$ then checks if flow x is a false positive by issuing a query to all nodes that had sent triggers to it for that flow; and if not, it raises an alarm for having detected an anomaly.

Although the trigger logic in CoCop(push) is deterministic as opposed to the probabilistic approach that CoCop(global) and CoCop(scoped) took, but they all use the *proportional* approach (see section 3) in which the communication overheads of the system are tied to the aggregate activity level in the system.

7.1.2 System Components

Each node has a *monitor* which maintains the flow counters and sends off triggers when required; and an *enforcer* that has the same functionality as in the pull-based system.

A critical architectural design decision is how to choose the coordinating nodes. A straight-forward approach is to have a single, central node that acts as the coordinating node for all flows, but this could become the central point of failure or a overload/congestion hotspot. An alternative is to leverage a distributed hash table (DHT) to spread the responsibility among all the participating nodes. The coordinating node for flow x, $N(x)$, could then simply be the DHT node corresponding to HASH(x). This, however, has the overheads associated with the maintenance of DHTs.

7.2 Analysis

Consider a flow x and let the aggregate traffic on it, $\sum_i f_i^x$, be F^x bps. Then, with triggers being sent for every $C\tau/n$ bits of traffic, a maximum of $n \cdot F^x/C\tau$ triggers can be sent to the coordinating node every second. The total communication overhead of the system is then bounded above by the summation of this expression over all flows, $\sum_x n \cdot F^x/C\tau \cdot 64$, where the factor of 64 is to account for the packet size of triggers and the conversion of units. Then we have:

$$\text{(overhead)} \quad \leq \quad n \cdot F/C \cdot \frac{64}{\tau} \tag{9}$$

Notice the similarity with the expression for overhead of CoCop(global) (see equation 4). This is because both

mechanisms follow the same approach of proportional triggering.

To calculate excess traffic, consider an anomalous flow x. It would take approximately $C\tau/F^x$ seconds for the coordinator, $N(x)$, to receive n triggers [1] . Then the excess traffic, which is the product of the excess rate, $(F^x - C)$, and the time-to-detection, is approximately:

$$\text{(excess traffic)} \quad \approx \quad C/F^x \cdot (F^x - C)\tau \cdot \frac{1}{8} \quad (10)$$

This is very low for F^x slightly greater then C and bounded from above by $C\tau$ for large F^x, *i.e.*, the excess traffic is bounded by a constant no matter whether the anomaly is small or large.

7.3 Simulations

7.3.1 Methodology

For simulating CoCop(push), we need traces that log the timestamp everytime a node sends on a flow its fair-share worth of traffic, $C\tau/n$ (and hence resets the counter and sends a trigger for that flow). However, as mentioned in section 6.1, the traces that we collected on PlanetLab record at a 1-second granularity the volume of traffic that each node sends on each flow; and thus, hide any information about the burstiness of traffic during that one second. So, to obtain traces for simulating CoCop(push) from the our PlanetLab traces, we make a simplifying approximation that the traffic that a node sent on a flow during a second, was sent at a uniform rate over that one second. For example, even if entire volume of traffic sent to a destination was in a burst lasting 20ms, as a result of using this approximation, in the traces used in simulating CoCop(push), it would appear as if the volume was sent at a uniform rate over one second.

It is easy to see that this approximation could give rise to both false positives and false negatives. However, a simple experiment (that we now describe) on actual traces shows that performance results change very marginally because of this approximation. Simulations were run over two different traces. The 1-second granularity traces were "grouped" to get 10-second granularity traces and then *Trace-A* was obtained using the above simplifying approximation. *Trace-B* was "closer to reality" in that it was obtained by directly using the 1-second granularity traces. Simulations of CoCop(push) showed little difference in the communication overhead, excess traffic or accuracy between these two traces.

[1] Though there are pathological (worst) cases where only $n \cdot (F^x/C - 1)$ triggers are received over τ seconds inspite of flow x being anomalous, *i.e.*, $F^x > C$.

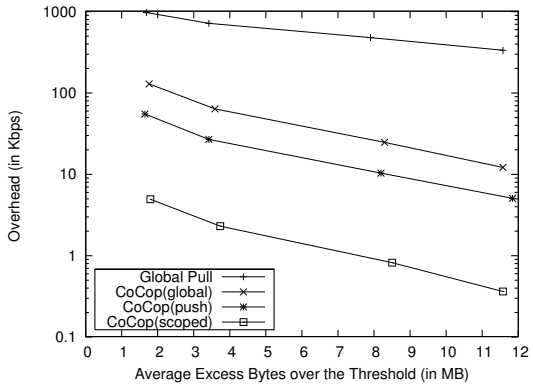

Figure 10: Tradeoff between Average Excess Traffic and Communication Overhead. The curve of CoCop(push) lies about below CoCop(global). CoCop(scoped) is still the lowermost curve.

We now present the performance results from simulation of CoCop(push) and compare it to the pull-based mechanisms.

7.3.2 Performance

Similar to the pull-based mechanisms, simulations of CoCop(push) show that by increasing the parameter τ, communication overhead decreases and the average excess traffic increases. And this should come as no surprise considering that the analysis also suggested so. What is interesting though, is the placement of its trade-off curve relative to other mechanisms. Figure 10 plots the (log of) communication overhead versus the excess traffic for CoCop(push) along with curves of periodic querying and pull-based CoCop from Figure 7.

CoCop(push) performs better than CoCop(global) on the PlanetLab traces, as is illustrated by the fact that the curve for CoCop(push) is lower and to the left of the curve for CoCop(global). Infact for the same average excess traffic, the former has only about half the communication overhead. This tallies well with the analytical bounds where the expression for overheads of these mechanisms differ by a factor of 2. CoCop(scoped), however, is still the lowermost curve, illustrating its a win-win choice over the rest.

In section 6.3.2, we studied the accuracy of mechanisms *relative* to periodic querying. This is because the best a pull-based mechanism can do (in terms of accuracy), is to catch all the anomalies that periodic querying does. However, as discussed in that section, there still are "true" anomalies (as defined by equation (1)) in the traces that are not detected even by periodic querying because of its hopping-window semantics. But these can (and in

most cases, *are*) detected by CoCop(push) because of the sliding-window semantics that the coordinating node uses. Therefore, to compare CoCop(push) with periodic querying and the two pull-based CoCop variants, Table 3 shows their (absolute) accuracy.

Trigger Interval	1s	2s	5s	10s
Periodic Query	1.0000	0.9957	0.9883	0.8823
CoCop(global)	1.0000	0.9822	0.9876	0.8627
CoCop(push)	0.9929	0.9884	0.9801	0.9807
CoCop(scoped)	0.9660	0.9767	0.9620	0.8627

Table 3: Accuracy of different mechanisms. CoCop(push) is about as accurate as CoCop(global) except at large trigger intervals where it is much better.

CoCop(push) is about as accurate as CoCop(global), except for the trigger interval of 10s where it is much more accurate than even periodic querying (for the reason just described). Although there are anomalies which only CoCop(push) detects, there are others that pull-based mechanisms detect but CoCop(push) doesn't. This is because, at each node, there could always be (just less than) a node's-fair-share-worth of traffic, ($C\tau/n$), for which no trigger is sent. This could lead to an under-estimation of the aggregate value by the coordinating node and hence lead to a false-negative.

8 Discussion

In the previous sections, we have seen several mechanisms to detect anomalous aggregates. Our efforts to taxonomize these, [15], suggest that there is still a large, unexplored design space of such mechanisms. Not only could the algorithms themselves be tweaked (especially the push-based mechanisms, which are far from being fleshed-out), but more fundamental changes could be made to the system design and architecture itself. There is also the possibility of hybrid mechanisms that use different combinations of variable-settings at different scales.

But what we hope the design and evaluation of these mechanisms have illustrated is that it is indeed possible to design mechanisms that can detect anomalous aggregates more rapidly and with significantly (orders-of-magnitude) lesser overheads than the current de facto approach of periodic querying. Such mechanisms raise the possibility of new kinds of monitoring for large, distributed systems that simultaneously check much larger number of conditions than were previously possible, or enable more open systems.

Although CoCop(scoped) emerged as the clear winner in the simulations on our PlanetLab traces, it would be presumptuous to claim that it should be the algorithm of choice whenever one needs to detect any number of anomalies in any aggregate property of any large, distributed system. On the contrary, we believe different mechanisms have different properties and could be the one best-suited to some scenario.

For small distributed systems having few global constraints that need to be monitored, periodic querying could be a pragmatic choice by virtue of its simplicity and clear semantics.

At the very other extreme, for very large distributed systems with thousands of nodes, a synchronous pull from all the nodes becomes infeasible and push-based schemes provide the only alternative.

It is medium-scale distributed systems that puts one a slippery ground. One major architectural decision that has to taken with regards a push-based design is to choose the "nodes that coordinate for a flow". We proposed the two alternatives of having one central coordinator or of having a DHT substrate, and discussed issues with each. Pull-based CoCop variants steer clear of this issue by not anointing any such canonical node. What they do require is the knowledge of other participating nodes - which is easy to maintain in a fairly static system such as PlanetLab, but might be overwhelming in a system with high churn. Choosing between CoCop(global) and CoCop(scoped) is to an extent driven by performance expectations from the system. CoCop(scoped) trades-off slightly higher average excess and a little lower accuracy for significantly lower overhead.

PlanetLab is fairly-static, large-scale distributed system, which is still small enough to support synchronous pull from all nodes (as our real experiments on PlanetLab have shown). While we ideally want to catch all anomalous aggregates, but we do not have very strict accuracy expectations and are not particularly concerned about missing those that are just above the threshold. All these factors make CoCop(scoped) well-suited to the scenario of detecting anomalous flow aggregates on PlanetLab. But we reiterate that in a different setting, another mechanism could be better-suited.

9 Related Work

Recently, there have been many proposals for large scale network monitoring systems such as PIER [14], Sophia [34], SDIMS [36], IrisNet [27] that along with monitoring individual nodes of the system, also monitor their aggregate behavior as a whole. While these all are well-suited when one wants to continuously monitor global state or obtain an instantaneous snapshot, none of

them were specifically designed to detect anomalies in these aggregates. Checking for anomalies is achieved by *periodically* polling the aggregate values - which are accumulated at a location (a node [14] or within the network [34]) either using a coupled pull [14, 34, 27] or decoupled push [36]. Section 6, however shows that our approach of *proportional* querying outperforms *periodic* querying for the specific task of detecting anomalous aggregates.

There is significant work in the networking community [39, 9, 13] and others([26, 18]) on techniques that can be used to find the elephants flows (flows with value above a prespecified threshold), heavy-hitters (flows that account for atleast a specified proportion of the total activity) and the top-k flows in a traffic stream. Most of this work has focused on a single data stream. The problem that we solve essentially reduces to finding elephants (also called "iceberg queries") in a distributed data stream.

Gibbons and Tirthapura introduced the notion of distributed data streams in [11], extended it to sliding windows in [12], and gave algorithms to find the bitwise OR of the streams and the number of distinct values in the distributed streams. Perhaps the work closest in spirit to ours is the distributed top-k monitoring mechanism proposed by Babcock et al [3], which continuously monitors the top-k aggregates. Using it to solve our problem of "aggregates above a threshold" however, leads to needless overheads. In the common case when there is nothing anomalous in the system, all the top-k aggregate flows would be sending at a rate less than the threshold. Moreover, it is quite likely that the top-k aggregate flows will keep on changing, leading to needless exchange of control traffic.

There is an immense amount of work on DDoS prevention and defense mechanisms. To the best of our knowledge this work assumes support from either the destination (*e.g.*[30]) or the network itself(*e.g.*, [32, 23]). Those that tackle it at source D-WARD [25] do not aggregate information across a set of nodes. Nobody in our knowledge has tried to look at preventing DDoS at sources (either cooperating or those belonging to a large open distributed system) by looking at flow aggregates.

Complementary to our work is the rich literature on computing aggregates over large distributed systems: using multiple aggregation trees [4], gossiping [31, 19, 16], hierarchy [31] and model-based acquisition [6], etc. Model-driven data acquisition suits well when, unlike in our setting, there is high temporal or spatial correlation. Aggregation has been studied in the context of continuous monitoring of sensor networks domain using querying [22] and directed diffusion [40]. It would be an interesting direction for future work to adapt CoCop for the sensor network domain where communication is also at a premium and detecting anomalies in aggregates, rather than continuously monitoring them, is quite often all that is needed.

We highlight some analogous themes in databases research. Standard database triggers, which form the basis of "active databases" [35], tend to focus on matching(joining) events, rather than on aggregates. In continuous queries over distributed stream query system [29, 22], dataflow moves either on each data arrival, or periodically, which ever comes first; and ostensibly we can do better than either in our setting.

There have been many results over the years in *communication complexity* [20] which studies the amount of information that needs to be communicated between parties that wish to reach a common computational goal. The two-party model was introduced by Yao in [37] and has since been extended in several direction - multiparty protocols [5], deterministic vs. randomized [38], public vs private random coins [28], etc. Our protocol can best be classified as a "randomized multiparty protocol with private coins". Although such protocols have been used to compute other aggregate functions such as basic boolean functions [7], we are not aware of any work that addresses the aggregate function in our problem.

10 Conclusions

In this paper, we tackle the problem of quickly and efficiently detecting anomalous aggregates in a large, distributed system. We describe a lightweight approach that makes the system-wide collection of data to detect anomalies proportional to the level of system activity, and a system, *CoCop*, that is based on it.

We evaluate CoCop with a trace-driven simulation that uses six days of detailed PlanetLab-wide flow data. We find that CoCop can detect anomalous aggregates as quickly as other methods, detects widely distributed anomalies as easily as concentrated ones, and has orders-of-magnitude lower communication overhead. Further, all of the algorithms we study exhibit a tradeoff between communication and the excess traffic before the anomaly is detected, but CoCop is able to outperform them at all operating points. We also derive bounds for communication overhead and excess traffic and find them to be in good agreement with our experimental results.

There is still a large, unexplored design space, but CoCop illustrates that it is indeed possible to design mechanisms that can detect aggregate anomalies more rapidly and with significantly lesser overheads than periodic query-

ing. Such rapid and efficient detection mechanisms raises the possibility of new kinds of monitoring for large, distributed systems that simultaneously check much larger numbers of conditions than were previously possible. In the future we plan to explore combinations of distributed triggers with other algorithms for locally detecting anomalies [8, 21], as well as other kinds of aggregate properties that may be useful for distributed intrusion detection.

References

[1] Planetlab: An open platform for developing, deploying and accessing planetlary-scale services.

[2] ulogd: Userspace packet logging for netfilter.

[3] B. Babcock and C. Olston. Distributed top-k monitoring, 2003.

[4] M. Bawa, H. Garcia-Molina, A. Gionis, and R. Motwani. Estimating aggregates on a peer-to-peer network. Tech. rep., Computer Science Department Stanford University, 2003.

[5] A. K. Chandra, M. L. Furst, and R. J. Lipton. Multi-party protocols. In *Proc. of the 15th annual ACM STOC, p.94-99*, 1983.

[6] A. Deshpande, *et al.* Model-driven data acquisition in sensor networks. In *Proc. of VLDB*. Toronto, 2004.

[7] D. Dolev and T. Feder. Determinism vs. nondeterminism in multiparty communication complexity. *SIAM J. Comput.*, 21(5):889–895, 1992.

[8] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *SIGCOMM '03*, pp. 137–148, 2003.

[9] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *Proc. of the SIGCOMM 2002*, 2002.

[10] S. Floyd, *et al.* A reliable multicast framework for lightweight sessions and application level framing. In *Proc. of the SIGCOMM 1995*. ACM Press, 1995.

[11] P. B. Gibbons and S. Tirthapura. Estimating simple functions on the union of data streams. In *ACM SPAA*, 2001.

[12] P. B. Gibbons and S. Tirthapura. Distributed streams algorithms for sliding windows. In *ACM SPAA*, 2002.

[13] L. Golab, *et al.* Identifying frequent items in sliding windows over on-line packet streams. In *Proc. of the 3rd ACM SIGCOMM IMC*, 2003.

[14] R. Huebsch, *et al.* Querying the internet with pier. In *Proc. of VLDB*, 2003.

[15] A. Jain, J. M. Hellerstein, S. Ratnasamy, and D. Wetherall. A Wakeup Call for Internet Monitoring Systems: The Case for Distributed Triggers. In *Proc. of the HotNets-III, San Diego, CA, USA*, 2004.

[16] M. Jelasity, W. Kowalczyk, and M. van Steen. An approach to massively distributed aggregate computing on peer-to-peer networks. In *Proc. of the 12th Euromicro PDP'04*. A Coruna, Spain, 2004.

[17] B. Karp, S. Ratnasamy, S. Rhea, and S. Shenker. Adoption of dhts with openhash, a public dht service. In *Proc. of the 3rd International Workshop on Peer-to-Peer Systems*, 2004.

[18] R. M. Karp, S. Shenker, and C. H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *ACM Trans. Database Syst.*, 28(1), 2003.

[19] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. In *Proc. of FOCS*, 2003.

[20] E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge, 1996.

[21] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies, 2004.

[22] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: A Tiny AGgregation service for ad-hoc sensor networks. In *OSDI '02*. Boston, 2002.

[23] R. Mahajan, *et al.* Controlling high bandwidth aggregates in the network. *SIGCOMM Comput. Commun. Rev.*, 32(3):62–73, 2002.

[24] A. Manjhi, V. Shkapenyuk, K. Dhamdhere, and C. Olston. Finding (recently) frequent items in distributed data streams. In *To appear in ICDE*, 2005.

[25] J. Mirkovic, G. Prier, and P. L. Reiher. Attacking ddos at the source. In *Proc. of ICNP '02*, 2002.

[26] S. Muthukrishnan. Data streams: Algorithms and applications.

[27] S. Nath, *et al.* Irisnet: An architecture for internet-scale sensing services. In *VLDB*, 2003.

[28] I. Newman. Private vs. common random bits in communication complexity, 1991.

[29] C. Olston, J. Jiang, and J. Widom. Adaptive filters for continuous queries over distributed data streams. In *ACM SIGMOD International Conference on Management of Data*, pp. 563–574. San Diego, 2003.

[30] C. Papadopoulos, *et al.* Cossack: Coordinated suppression of simultaneous attacks.

[31] R. V. Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2), 2003.

[32] S. Savage, D. Wetherall, A. R. Karlin, and T. Anderson. Practical network support for IP traceback. In *SIGCOMM*, 2000.

[33] N. Spring, D. Wetherall, and T. Anderson. Scriptroute: A public internet measurement facility, 2002.

[34] M. Wawrzoniak, L. Peterson, and T. Roscoe. Sophia: An Information Plane for Networked Systems. In *Proc. of the HotNets-II, Cambridge, MA, USA*, 2003.

[35] J. Widom and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann, San Francisco, 1996.

[36] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *Proc. of the SIGCOMM 2004*. ACM Press, 2004.

[37] A. C.-C. Yao. Some complexity questions related to distributive computing. In *Proc. of the 11th annual ACM STOC, p.209-213*, 1979.

[38] A. C.-C. Yao. Lower bounds by probabilistic arguments. In *Proc. of IEEE FOCS, p.420-428*, 1983.

[39] Y. Zhang, *et al.* Online identification of hierarchical heavy hitters: Algorithms, evaluation and application. In *Proc. of the ACM SIGCOMM IMC*, 2004.

[40] J. Zhao, R. Govindan, and D. Estrin. Computing aggregates for monitoring wireless sensor networks, 2003.

APPENDIX

Given that out of n nodes, $(F^x/C)k$ are at C/k and the rest at zero; and node i has decided to initiate a scoped query. What is the (lower-bound on the) probability q_i^x of *success* (*i.e.*i will find that the sum of all nodes is $> C$)?

Observe that if, instead of $(F^x/C)k$, we consider k nodes at C/k, then q_i^x can only reduce as the number of contributors has reduced. Also, for ease of argument, assume n to be the smallest power of 2 greater than n.

Now consider the step at which node i has already asked $n/2$ nodes. It will proceed on to ask the remaining n nodes, iff there were $>= k/2$ contributors in the set of nodes it had already asked. Now, if we look at this step independent from all the previous steps, the probability that the $n/2$ nodes it has asked has atleast $k/2$ contributors is 1/2. Note that the independence assumption will only reduce the probability. This is because, if i reaches this step, then it must have been successful in the previous step also and hence would have already asked atleast $k/4$ contributors. By assuming independence, we are ignoring this prior, which would have increased the probability. Similarly, consider the second last step at which node i has already asked $n/4$ nodes. It will proceed to ask a total of $n/2$ (including the current $n/4$) iff there were $>= k/4$ contributors in the set of nodes it had asked. Like before, the probability for this 1/2. Combining this with the last step, the probability of success from this step is $(1/2)^2$. In general, from the j-th last step, the probability of success is $(1/2)^j$.

Since node i itself was as C/k, it would have started by asking $2^{\log\lceil n/k \rceil}$. Thus, the maximum number of steps that i can take are $\log k$. Hence, $q_i^x >= (1/2)^{\log k} = 1/k$.