A Mathematical Framework for the Detection of Elephant Flows

Jordi Ros-Giralt, Alan Commike
Reservoir Labs
632 Broadway Suite 803, New York, NY 10012

Sourav Maji, Malathi Veeraraghavan
Dept. of Electrical and Computer Eng., University of Virginia Charlottesville, VA 22904–4743

Abstract
How large is a network flow? Traditionally this question has been addressed by using metrics such as the number of bytes, the transmission rate or the duration of a flow. We reason that a formal mathematical definition of flow size should account for the impact a flow has on the performance of a network: flows that have the largest impact, should have the largest size. In this paper we present a theory of flow ordering that reveals the connection between the abstract concept of flow size and the QoS properties of a network. The theory is generalized to accommodate for the case of partial information, allowing us to model real computer network scenarios such as those found in involuntary lossy environments or voluntary packet sampling protocols (e.g., sFlow). We explore one application of this theory to address the problem of elephant flow detection at very high speed rates. The algorithm uses the information theoretic properties of the problem to help reduce the computational cost by a factor of one thousand.

1. Introduction
A general objective in the design of high-performance computer networks is to guarantee the quality of service (QoS) experienced by the data flows that traverse them. This objective is often challenged by the presence of very large flows—also known as elephant flows—due to their adverse effects on smaller delay-sensitive flows. Because in these networks both large and small flows share common resources, network operators are interested in actively detecting elephant flows and using QoS mechanisms for redirecting and scheduling them to protect the smaller flows.

The problem of elephant flow characterization and detection has been the subject of intense research for the last fifteen years. Since then, a considerable amount of work has focused on the problem of identifying key flow metrics such as byte counts, rate or burstiness [1, 2, 3, 4, 5] to help identify when a flow ought to be classified as an elephant. Metric-centric approaches however offer locally optimal solutions as they are agnostic to the general QoS requirements of the network. For instance, it’s logical to think that a flow with a high bytecount should not be classified as elephant if no other flow’s QoS is affected by it, regardless of how high its bytecount is. Related to this subject but of more general interest is attempting to formally resolve the problem of flow ordering: given a set of flows sharing a common network, which is the flow that incurs the n-th largest impact on the performance of a network? Further, the general problem of flow ordering has a more fundamental unresolved issue: its formulation presumes the existence of the concepts of flow ordering and flow size, for which we lack a formal mathematical definition.

To address these issues, in the first part of this paper we present a new theoretical framework that leads to the definition of the abstract concept of flow ordering and flow size in connection with the global quality of service (QoS) requirements of a network. This framework demonstrates that the problem of elephant flow detection is a particular case of a more general problem which we call the partitioned QoS problem, and present the convexity properties that ensure the flow ordering is well-defined, computationally tractable, and with a valid metric. We also demonstrate that metric-based approaches found in the existing literature are the solution to one particular class of network problems, and that a mathematically correct definition of flow size can only be constructed from the partitioned QoS problem. Then, we show that if a partitioned QoS problem is nested convex as formally defined in this work, then it can be solved in polynomial time.

The mathematical framework provides a base theory of flow ordering without uncertainty. Real practical networks, however, need to operate under uncertainty or partial information. Sources of uncertainty can come from either a natural inability to predict the traffic’s future performance or from artifacts introduced by networking equipment such as involuntary packet drops or voluntary packet sampling from protocols like sFlow [6]. In the second part of this paper, we focus on the problem of flow ordering under uncertainty. The problem of identifying the minimum amount of information needed to detect the largest flows in a network is addressed. Then, under the assumption of heavy tailed traffic, we demonstrate the existence of cutoff sampling rates. Similar to the concept of Nyquist sampling rate in signal processing, the cutoff sampling rate of a heavy tailed traffic dataset corresponds to the minimum rate at which traffic must be sampled in order to detect and reconstruct the top flows with high probability. The theory provides exact formulas to compute the detection likelihood, a key building block to design packet sampling
algorithms operating near the optimal tradeoff between computational scalability and accuracy. It also leads to the flow reconstruction lemma, which states that if the sampled traffic dataset is heavy tailed, then the detection system operates error free with high probability.

In the final part of this paper, we use the theory of flow ordering under uncertainty to design the BubbleCache algorithm, a high performance flow cache algorithm that retains the top flows by dynamically tracking the optimal cutoff sampling rate. We demonstrate on a real world 100 Gbps network that the BubbleCache algorithm can help reduce the computational cost by a factor of 1000 and the memory requirements by a factor of 100 while detecting the largest flows on the network with high probability. Two direct applications of the BubbleCache algorithm are the design of optimal packet sampling modules such as those used in protocols like sFlow [6] and the design of high performance queues to dynamically separate elephant and mouse flows and to protect them from each other. More in general, the theory of flow ordering presented in this work can be used to provide operators with a framework to diagnose and optimize network QoS.

2. Theory of Flow Ordering Under Certainty

2.1. General Definition of Elephant Flow

Let us initiate our theoretical framework with a definition: Definition 1. Partitioned QoS function. Let $F$ be a set of flows transmitting data over a network $N$. Assume that each flow is assigned resources from network $N$ according to one of $l$ possible policies $P_1, P_2, \ldots, P_l$. Let the tuple $<F_1, F_2, \ldots, F_l>$ be a network configuration formed by mutually exclusive sets whose union is $F$. We will say that $q(F_1, F_2, \ldots, F_l)$ is an $l$-partitioned QoS function of network $N$ on the set of flows $F$ if $q(F_1, F_2, \ldots, F_l) \geq q(F'_1, F'_2, \ldots, F'_l)$ implies that the quality of service achieved with configuration $<F_1, F_2, \ldots, F_l>$ is greater than with configuration $<F'_1, F'_2, \ldots, F'_l>$. When there is no need to make the number of partitions explicit, we will also say that $q()$ is a partitioned QoS function or simply a QoS function.

A partitioned QoS function is therefore a utility function on the space of all the possible $l$-partitions of $F$. Throughout this essay, the following encoding will provide a convenient way to express some of the mathematical and algorithmic results:

Definition 2. Configuration encoding. A network configuration expressed as an $l$-partition of a set of flows $F$, $<F_1, F_2, \ldots, F_l>$, can be conveniently encoded as an $|F|$-dimensional vector $p$ where each element $p_i$ is such that:

$$p_i = k \text{ if } f_i \in F_k$$

Hence, any $l$-part partition has one corresponding vector in the space $[1]^{|F|}$, where $|l| = \{k \in \mathbb{N} | k \leq l\}$. Using this encoding, a partitioned QoS function can be conveniently expressed as a utility function mapping the set of vectors in $[1]^{|F|}$ onto the set of real numbers $\mathbb{R}$:

$$q : [1]^{|F|} \rightarrow \mathbb{R} \text{ } \S$$

A partitioned QoS function can also be expressed in terms of finding ways to store $|F|$ objects into $l$ boxes according to a utility function $q()$. In this case, the search space is given by the sum of all multinomial coefficients, which corresponds also to the size of the space $[1]^{|F|}$:

$$\sum_{k_1, \ldots, k_l \in [1]} |F| = |F| \text{ } \S$$

The problem of finding an optimal network configuration $<F_1, F_2, \ldots, F_l>$ can now be reduced to the problem of identifying the element in $[1]^{|F|}$ that maximizes $q()$:

Definition 3. Partitioned QoS problem. We will say that an $l$-part partition $<F_1, F_2, \ldots, F_l>$ of $F$ is a QoS optimal partition if $q(F_1, F_2, \ldots, F_l) > q(F_1, F_2, F_i)$ for any other $l$-part partition $<F_1, F_2, \ldots, F_i>$ of $F$. We will refer to the problem of finding such optimal partition as the partitioned QoS problem or the $l$-partitioned QoS problem if there is a need to make the number of partitions explicit.

This formulation leads to a formal definition of elephant and mouse flows:

Definition 4. Elephant and mouse flows. Let $F$ be a set of flows transmitting data over a network $N$. Assume that each flow is assigned resources from network $N$ according to two possible policies, $P_1$ and $P_2$, and let $q()$ be a 2-partitioned QoS function. Then we will say that $F_e \subseteq F$ is the set of elephant flows if and only if $<F_e, F \setminus F_e>$ is a QoS optimal partition. Consequently, we will say that $F \setminus F_e$ is the set of mouse flows.

The above definition shows that the problem of identifying the set of elephant flows in a network corresponds to an $l$-partitioned QoS problem with $l = 2$. It also allows us to establish a direct relationship between the meaning of elephant flow and its effects on the QoS of a network: the set of elephant flows is one that when exclusively assigned to policy $P_1$, the resulting QoS of the network is maximal. The same can be said of mouse flows: the set of mouse flows is one that when exclusively assigned to policy $P_2$, the resulting QoS is also maximal. Notice that without loss of generality, we will take the convention $<F_1, F_2> = <F_e, F \setminus F_e>$, so that the set of elephant and mouse flows are assigned policies $P_1$ and $P_2$, respectively.
Following the notation in equation (1), any 2-partitioned QoS function $q()$ can be encoded using a mapping between the set of $|F|$-dimensional binary vectors and the set of real numbers:

$$ q : b^{[|F|]} \Rightarrow \mathbb{R} $$

(4)

where $b = [2] = \{0, 1\}$.

**Lemma 1.** Metric-based partitioned QoS. Let $m(f)$ be any function mapping the set of flows $F$ with the set of nonnegative real numbers, $m : F \Rightarrow \mathbb{R}^+$. (We will also call $m(f)$ a *metric function.*) Assume the following 2-partitioned QoS function:

$$ q(F_1, F_2) = \begin{cases} 
K_1 - \sum_{f \in F_2} \pi(f), & \text{if } \sum_{f \in F_2} r(f) > C \\
K_2 + \sum_{f \in F_2} \rho(f), & \text{otherwise}
\end{cases} $$

(5)

where $r(f)$ is the rate of flow $f$, $C$ is a network capacity parameter, $K_1$ and $K_2$ are two arbitrary positive numbers chosen so that $K_2 > K_1$, and $\pi(f)$ and $\rho(f)$ are two penalty and reward functions such that:

$$ \pi(f_i) > \pi(f_j) \iff m(f_i) > m(f_j) \quad \rho(f_i) > \rho(f_j) \iff m(f_i) < m(f_j), \text{ for all } f_i, f_j \in F $$

(6)

Then the set of elephant flows corresponds to $F_1 = F_e = \{ f_1, f_2, ..., f_{k_1} \}$ such that $m(f_i) > m(f_{i+1})$, $\sum_{i \in \lambda} r(f_i) \leq C$, and $F_e$ has minimal cardinality.

**Proof.** Equation (5) can be modeled using the network described in Figure 1 as follows. If the rate of traffic going through the high-priority queue (policy $P_2$) exceeds the total capacity of the network, $C$, the QoS of the network deteriorates according to the penalty function $\pi()$. Since from equation (6) flows with the highest metric $m()$ will be assigned the largest penalty, we can improve the QoS of the network by routing these high-metric flows to the low-priority queue. On the other hand, if the rate of traffic going through the high-priority queue (policy $P_2$) is below the total capacity of the link $C$, the total QoS of the network improves according to the reward function $\rho()$. In this case, the QoS of the network can be improved by migrating the flows with the lowest metric $m()$ from the low-priority queue to the high-priority queue.

The optimal 2-partitioned QoS is therefore one that redirects the minimum number of high-metric flows to the low-priority queue without exceeding the capacity of network, $C$. This partitioned QoS problem can be trivially solved as follows. Let $(f_1, f_2, ..., f_{l+1})$ be the list of flows ordered according to the rule $m(f_i) \geq m(f_{i+1})$. Then the set of elephant flows corresponds to $F_1 = F_e = \{ f_1, f_2, ..., f_{k_1} \}$ such that $\sum_{i \in \lambda} r(f_i) \leq C$ and $F_e$ has minimal cardinality—i.e., $\lambda$ is minimal.

The 2-partitioned QoS function in equation (5) demonstrates that for any elephant flow detection algorithm in the literature which uses a metric function to classify elephant flows, there exists at least one partitioned QoS function whose solution leads to the ordering given by the metric. For instance, elephant flow definitions such as those based on the metrics of flow rate [2, 3, 4], bytecount [2, 3] or burstiness [2, 3, 5] are all equivalent to solving the partitioned QoS problem in Figure 1. This indicates that current solutions only address a specific type of QoS problems: those characterized by the partitioned QoS function in equation (5). Real-world network operators however need an architect and manage their networks according to arbitrary QoS requirements that depend on a large variety of factors—e.g., network topology, path latency, customer demand, network offering, etc.—which means that in practice their optimal solution will differ from the solutions offered by a metric-based QoS function. By leading us to a formal mathematical definition of flow size, the theory of flow ordering we introduce in this work will enable a new network optimization framework to support arbitrary QoS functions beyond metric-based solutions.

### 2.2. Computational Complexity

Equation (3) shows that the solution space of a partitioned QoS problem grows rapidly with both the number of flows $|F|$ and the number of policies available $l$. In this section we are interested in analysing the general computational complexity properties of the problem. This analysis will be helpful later on when we attempt to identify the properties of the partitioned QoS function $q()$ that make the partitioned QoS problem tractable. In what follows, we utilize standard notation from computational complexity theory. (See for instance [7].)

**Definition 5.** Abstract partitioned QoS Problem: $\text{QS}$. We define the abstract partitioned QoS problem, denoted by $\text{QS}$, as a binary relation on a set $I(\text{QS})$ of problem instances and a set $S(\text{QS})$ of problem solutions, where:

- A problem instance is a tuple $< N, F, P_1, P_2, ..., P_l, q() >$ consisting of a network $N$, a set of flows $F$, a set of $l$ policies $P_1, P_2, ..., P_l$ and a utility function $q : [l]^{|F|} \Rightarrow \mathbb{R}$. 

![Figure 1. Equivalent network for metric-based QoS problems](image)
- A solution instance is one optimal QoS partition $<F_1^*, F_2^*,..., F_n^*>$. §

In computational theory, abstract problems are usually expressed in terms of decision problems [7]. The following definition introduces the decision problem associated to QoS:

**Definition 6. Decision partitioned QoS Problem: QSD.** Let Q5 be an abstract partitioned QoS problem and let $k \in \mathbb{R}$ be an arbitrary real number. We define the decision partitioned QoS problem, denoted by QSD, as a binary relation on the set $I(QSD) = I(Q5)$ to the solution set \{0, 1\} such that:

$$QSD(<N,F,P_1,F_2,...,P_n,q()> = 1 \text{ if } q(F_1,F_2,...,F_n) \geq k \text{ and } \quad (7)$$

$$QSD(<N,F,P_1,F_2,...,P_n,q()> = 0 \text{ otherwise. } \quad \text{§}$$

We can now state the general complexity of QSD problems:

**Lemma 2. General complexity.** The set $I(QSD)$ includes P, NP and NP hard problems. That is, $I(QSD) \subset NP$ and $I(QSD) \cap P = \varnothing$.

**Proof.** See the Appendix for a formal proof. §

Since in our work the set of QoS problems dealing with two partitions will be of special interest, we provide a dedicated definition:

**Definition 7. Abstract and decision problems for the 2-partition configurations: Q52 and QSD2.** We define Q52 and QSD2 as the set of abstract and decision partitioned QoS problems, respectively, such that $l = |P| = 2$. §

### 2.3. General Definition of Flow Ordering

We now center around the problem of deriving a formal definition of flow size and ordering for computer networks. While we will discuss this subject within the space of problems in Q52, the following results are generally applicable to problems in Q5 as shown in the Appendix.

**Definition 8. $\lambda$-optimal QoS partition.** Let $<N,F,P_1,F_2,...,P_n,q()> \in Q52$ be a 2-partitioned QoS problem. We will say that $F_\lambda \subseteq F$ defines a $\lambda$-optimal QoS partition on the set of flows $F$ if $q(F_\lambda, F \setminus F_\lambda) \geq q(F_i, F \setminus F_i)$ for all $F_i$ such that $F_i \subseteq F$ and $|F_\lambda| = |F| = \lambda$. When the meaning is clear, we will simply use the term $\lambda$-optimal partition. §

Notice that using definitions 4 and 8, the set of elephant flows $F^e$ is an $|F^e|$-$\lambda$-optimal QoS partition. Definition 8 reveals also a recursive strategy to identify the ordering of flows according to an abstract notion of size as follows:

- Start by finding $F_1 = \{f_1\} \subseteq F$ such that $F_1$ is a 1-optimal partition. Mark $f_1$ as the largest flow.

- Next, find $F_2 = \{f_1, f_2\} \subseteq F$ such that $F_2$ is a 2-optimal partition. Mark $f_2$ as the second largest flow.

- Continue recursively for all possible $\lambda$-optimal partitions until all the flows are marked.

Notice that in order for the above construction process to succeed, the condition $F_\lambda \subseteq F_{\lambda + 1}$ must be satisfied at each iteration. This leads to the first property that a 2-partitioned QoS function needs to satisfy to ensure the flows can be ordered according to their size:

**Property 1. Inclusive QoS functions.** We will say that a 2-partitioned QoS function is inclusive if its $\lambda$-optimal partition $F_\lambda$ includes the $(\lambda - 1)$-optimal partition $F_{\lambda - 1}$, for $2 \leq \lambda \leq |F|$. Equivalently, assuming the vector $p_\lambda$ is the configuration encoding of $<F_\lambda,F \setminus F_\lambda>$ (Definition 2), a 2-partitioned QoS function is inclusive if $p_\lambda = p_{\lambda - 1} \land p_{\lambda}$, where $\land$ is the bitwise logical conjunction. §

We are now in a position to formally introduce the concept of flow size ordering:

**Definition 9. Flow size ordering.** Let $<N,F,P_1,F_2,...,P_n,q()> \in Q52$ be a 2-partitioned QoS problem and assume that $q()$ is inclusive. We will say that a flow $f_i \in F$ is the $i_{th}$ largest flow in the set $F$ if and only if:

$$q(F_i, F \setminus F_i) \geq q(F_j, F \setminus F_j)$$

where:

$$F_i = F_{i-1} \cup \{f_i\}, \quad f_i \notin F_{i-1},$$

$$F' = F_{i-1} \cup \{f'\} \text{ for all } f' \notin F_{i-1} \text{ and } F_0 = \{0\}$$

We will also say that the ordered list $\{f_1, f_2, ..., f_{|F|}\}$ defines the flow size ordering of the 2-partitioned QoS problem $<N,F,P_1,F_2,...,P_n,q()>$. §

The previous definition formally introduces the mathematical ordering of a set of flows according to their impact to the overall QoS of the network. In the particular case of a problem in the Q52 set, the top flows in this ordering correspond to the elephant flows.

While knowing the flow ordering is often enough to diagnose and optimize network problems, sometimes it can be useful to have an exact measurement or metric for the size of a flow. We construct such metric next.

**Property 2. Decreasing returns to gains.** Let $\{f_1, f_2, ..., f_{|F|}\}$ be the flow size ordering of a 2-partitioned QoS problem $<N,F,P_1,F_2,...,P_n,q()>$ and let $\sigma_q(f_i) = q(N,F,P,F_i) - q(N,F,F_i,F_{i-1})$. We will say that $q()$ has decreasing returns to gains, if the following is true:

$$\sigma_q(f_i) \leq \sigma_q(f_{i-1}) \text{, for } 2 \leq i \leq |F|$$

Intuitively, the property of decreasing returns to gains tells us that as more flows are moved into policy $P_1$, the QoS
gains achieved by performing such action become lower. This property leads to a natural definition of flow size as follows:

**Definition 10.** Flow size and q-size metric. Let \( \{f_1, f_2, ..., f_N\} \) be the flow size ordering of a QS2 problem \( <N,F,P_1,P_2, q()> \) and assume \( q() \) has decreasing returns to gains. We define \( \sigma(f_i) = q(F_i, F_i^{-}) - q(F_i^{-}, F_i^{-1}) \) as the size of flow \( f_i \). We will also refer to the function \( \sigma(f_i) \) as the q-size metric of the QS2 problem \( <N,F,P_1,P_2, q()> \). When the subindex \( q \) is redundant, we will simply use the expression \( \sigma(f_i) \) or \( \sigma_i \). §

We complete this analysis showing that all metric-based definitions of flow ordering have decreasing returns to gains and their true size is given by their penalty and reward functions:

**Lemma 3.** Decreasing returns to gains for metric-based ordering. All metric-based partitioned QoS functions (see Lemma 1) have decreasing returns to gains. Further, for these class of problems, the size of flow \( f_i \) is \( \sigma(f_i) = \pi(f_i) \) if it is an elephant flow and \( \sigma(f_i) = p(f_i) \) if it is a mouse flow.

**Proof.** See the Appendix. §

### 2.4. Inclusive QoS Functions

We are now interested in identifying the topological properties that partitioned QoS functions must satisfy to ensure that a flow ordering exists. To this end, we introduce the concepts of partition distance and nested neighborhood:

**Definition 11.** Partition and nested neighborhoods. Let \( <F_1, F_2> \) and \( <F_1', F_2'> \) be two 2-partitions of a set of flows \( F \) and let the vectors \( p \) and \( p' \) be their corresponding configuration encoding (Definition 2). We define the partition distance \( d_\pi() \) between \( <F_1, F_2> \) and \( <F_1', F_2'> \) as the difference between the number of flows assigned to \( F_1 \) and \( F_1' \). (Equivalently, it is also the difference between the number of flows assigned to \( F_2 \) and \( F_2' \).) Mathematically:

\[
d_\pi(p, p') = ||p||_1 - ||p'||_1
\]  

where \( ||x||_1 \) is the Manhattan norm. We also define the partition neighborhood \( n_\pi() \) of a 2-partition as the set of all 2-partitions that are at a partition distance 1 of it:

\[
n_\pi(p) = \{ p' \text{ s.t. } d_\pi(p, p') = 1 \}
\]  

The partition neighborhood allows us to define the concept of nested neighborhood \( n_\sigma() \) as follows:

\[
n_\sigma(p) = \{ p' \in n_\pi(p) \text{ s.t. } p = p' \wedge p' \}
\]  

where \( \wedge \) is the bitwise logical conjunction. §

We can now introduce the concept of nested convexity which will allow us to characterize the property of inclusiveness:

**Definition 12.** Nested convexity on 2-partitioned QoS functions. Let a configuration encoding \( p \) be a \( ||p||_1 \)-optimal partition of an arbitrary 2-partitioned QoS problem \( <N,F,P_1,P_2, q()> \in QS2 \). The QoS function \( q() \) is said to be nested convex in the vicinity of \( p \) if the following is true:

\[
\forall p' \in n_\pi(p) \exists p'' \in n_\sigma(p) \text{ s.t. } q(p'') \geq q(p') \quad (13)
\]

**Lemma 4.** Nested convexity and inclusiveness. A 2-partitioned QoS function is inclusive if and only if it is nested convex in the vicinity of its \( \lambda \)-optimal partitions, for all \( 1 \leq \lambda \leq |F| \).

**Proof.** Let \( <N,F,P_1,P_2, q()> \) be a 2-partitioned QoS problem and assume that \( q() \) is nested convex in the vicinity of its \( \lambda \)-optimal partitions but not inclusive. Then, from Property 1, there must be a \( \lambda \)-optimal partition which does not include the \( (\lambda - 1) \)-optimal partition, for some \( \lambda \) between 2 and \( |F| \). For such \( \lambda \), we have that \( P_{\lambda - 1} \neq P_{\lambda - 1} \wedge P_{\lambda} \) and, from equation (12), the following holds:

\[
P_{\lambda} \notin n_\sigma(P_{\lambda - 1})
\]  

Since \( P_{\lambda} \) is a \( \lambda \)-optimal partition, we also have that:

\[
q(P_{\lambda}) > q(p) \quad \text{for all } p \in n_\sigma(P_{\lambda - 1}) \quad (15)
\]

But equations (14) and (15) contradict the nested convex definition in equation (13). This proves the “if” part of the lemma. Refer to the Appendix for the “only if” part. §

We can now state the general complexity of the partitioned QoS problem for nested convex QoS functions:

**Lemma 5.** Computational complexity under nested convexity. Let \( <N,F,P_1,P_2, q()> \) be a 2-partitioned QoS function that is nested convex in the vicinity of all its \( \lambda \)-optimal partitions. Then \( <N,F,P_1,P_2, q()> \) belongs to \( P \).

**Proof.** See the Appendix. §

We conclude the theoretical framework summarizing in Figure 2 the possible configurations that a partitioned QoS problem can take. To be well-defined, the property of inclusive (equivalently, nested convex) must hold. This ensures both the existence of a flow ordering and that the problem belongs to \( P \). If in addition the property of decreasing returns to gain holds, then there exist a q-size metric.
3. Theory of Flow Ordering With Uncertainty

The first part of our theory demonstrated that the problems of flow ordering and elephant flow detection are a particular case of the partitioned QoS problem. This theory allowed us to establish (1) the mathematical connection between the size of a flow and the structural QoS properties of the network, (2) the general definition of the concepts of flow ordering and elephant flow and (3) the convexity properties that a QoS utility function must satisfy to make the problem well-defined and tractable. Now we notice that while these are fundamental concepts to help us approach the problem in a formal manner, they implicitly rely on the following assumption: that an observer of the network trying to identify the flow ordering has full knowledge of the network and all of its states. In particular, equation (8) assumes we can know the impact of each flow onto the overall QoS of the network with certainty. In practice, however, an observer trying to identify the flow ordering will not have full information of the system under observation at least because of two reasons:

- Future uncertainty. Unlike an oracle, we cannot predict the traffic that each flow will carry in the future and, hence, we cannot know with certainty the flow ordering.
- Past uncertainty. Even if we could predict the future states of a network, oftentimes networking equipment cannot keep up with the rates at which packets are processed in the data plane. For instance, in today’s networks, it is computationally expensive to monitor every single packet going through a 100 Gbps link. Under these conditions, packets often need to be sampled or dropped, adding another source of uncertainty.

As a result, any practical algorithm to order and detect the top flows needs to deal with uncertainty or partial information. In this second half of the theory, we focus on the problem of flow ordering and elephant flow detection assuming uncertainty and the design of practical high-performance algorithms to efficiently resolve these problems at very high-speed rates.

3.1. On The Effect of Sampling

Consider a simple initial problem with a traffic dataset consisting of one single flow carrying m packets and n flows carrying 1 single packet. Figure 3 displays the packet distribution corresponding to this traffic dataset.

Our interest is in finding a sampling strategy that allows us to identify the largest flow without necessarily processing all the traffic. To resolve this problem, we observe that if we sample two packets from the elephant flow, then we can assert with certainty which flow is the biggest, since none of the other flows have more than 1 packet. In particular, let X(k) be the number of packets sampled from the elephant flow out of a total of k samples taken from the traffic dataset. Then the probability of identifying the elephant flow with certainty is:

\[ P(X(k) \geq 2) = 1 - P(X(k) = 0) - P(X(k) = 1) \]  \hspace{1cm} (16)

Using combinatorics, it’s easy to see that:

\[ P(X(k) \geq 2) = \begin{cases} 
1 - \left( \frac{m}{m+n} \right)^2, & \text{if } 2 \leq k \leq n \\
1 - \left( \frac{m}{m+n} \right)^k, & \text{if } k = n + 1 \\
1, & \text{if } n + 1 < k \leq n + m
\end{cases} \]  \hspace{1cm} (17)

Figure 4 plots the above result for the case n = 1000, with m varying from 1 to 15 and with k = p(\(\frac{m+n}{p}\)), for 0 \leq p \leq 1. We notice that:

- For the boundary case \(m = 1\), the probability of finding the elephant flow is zero, since the elephant flow is indistinguishable from the small flows.
- As we increase the number of samples taken (p), the probability of finding the elephant flow increases.
- As the number of packets in the elephant flow increases (m), we need less samples to gain a higher probability of finding it.

![Figure 3. A simple heavy-tailed traffic dataset](image)

Of interest is to contrast the above result with the case of real world network traffic. It is well known that Internet traffic is characterized by heavy tailedness [9,14,17], a condition in which traffic consists of a small number of flows transmitting a very large amount of data and a large number of flows transmitting a small amount of data. As illustrated in our simple example, this natural characteristic of Internet traffic works in favor of detecting the elephant flows with high likelihood under partial information: a larger value of m, implies a higher degree of heavy tailedness, which leads to a higher likelihood to detect the elephant flow. We formalize this concept with a definition that will be useful later on when we address the problem of top flow detection under uncertainty:
Definition 13. Heavy tailed traffic. Let $F$ be a set of flows transmitting data over a network and assume $<F_e,F_m>$ is a QoS optimal partition of $F$ according to Definition 3—that is, $F_e$ and $F_m$ are the set of elephant and mouse flows, respectively, according to some 2-partitioned QoS function $q()$. Further, let $\sigma_i$ be the $q$-size metric of flow $f_i$ according to $q()$. We will say that the traffic dataset generated by the flows in $F$ is heavy tailed if $|F_e|<|F_m|$ and $\sigma_i >> \sigma_j$ for any pair of flows $f_i$ and $f_j$ in $F_e$ and $F_m$, respectively. Further, let $F'$ be another set of flows such that $<F_e',F_m'>$ is a QoS optimal partition of $F'$ and assume that $|F_e'|=|F_e|$ and $|F_m'|=|F_m|$. Assume also that flows are indexed in decreasing order of their size, $\sigma_i \geq \sigma_j$ and $\sigma'_i \geq \sigma'_j$ for any $i < j$. Then $F'$ is said to be more heavy tailed than $F$ if $\sigma'_i \geq \sigma_i$ and $\sigma'_j \leq \sigma_j$, for all $1 \leq i,j \leq |F|=|F'|$, and $\sigma'_i > \sigma_i$ or $\sigma'_j < \sigma_j$, for some $1 \leq i,j \leq |F|=|F'|$. §

![Figure 4. Probability to detect the top flow](image)

Our simple example in Figure 4 offers some initial insights on the problem of elephant flow detection under uncertainty but it comes with two limitations. First, the example deals with a very simple traffic dataset model consisting of 1 flow transmitting $m$ packets and $n$ flows transmitting 1 single packet. Second, it assumes that the size of a flow equates to its number of packets. As we know from the theory of flow ordering, this leads to implicitly assume a network model characterized by the metric-based QoS utility function in equation (5). However, the general definition of flow size for arbitrary QoS functions is provided by the $q$-size metric $\sigma(f_i)$ introduced in Definition 10.

In the next section, we derive a generalized expression of the likelihood to detect elephant flows for arbitrary traffic distributions and for definitions of flow size $\sigma(f_i)$ based on general QoS functions.

3.2. Generalization to Arbitrary Distributions

We start by introducing the definition of quantum error which will allow us to characterize the concept of detection likelihood under uncertainty.

Definition 14. Quantum error (QER). Let $F$ be a set of flows transmitting information over a network and let $x(t)$ be a vector such that its $i$-th element, $x_i(t)$, corresponds to the size of flow $i$ at time $t$ according to the $q$-size metric introduced in Definition 10. $x(t)$ is therefore a time-varying vector such that $x_i(t_b)=0$ and $x_i(t_e)=\sigma_i$, where $t_b$ and $t_e$ are the times at which the first and the last bit of information are transmitted from any of the flows, and $\sigma_i$ is the $q$-size metric of the flow at time $t_e$. Assume without loss of generality that $\sigma_i \geq \sigma_{i+1}$ and let $F_a = \{f_1,f_2,...,f_a\}$. Finally, let $C_a(t)$ be a cache storing the top $\alpha$ largest flows according to their size $x_i(t)$ at time $t$. (Hence, by construction, $C_a(t_e)=F_a$.) We define the quantum error (QER) produced by the cache at time $t$ as:

$$z_{\alpha}(t) = \frac{\sum_{\{x_i(t) s.t. \sigma_i < \sigma_{a} \text{ and } x_i(t) > x_{\alpha}(t)\}}}{\alpha}$$

(18)

Intuitively, the above equation corresponds to the number of small flows that at time $t$ are incorrectly classified as top flows normalized so that the error is 1 if all top $\alpha$ flows are misclassified. Because this error refers to the notion of an observer classifying a flow at an incorrect policy level, we use the term quantum error or QER. We can now formally introduce the concept of detection likelihood:

Definition 15. Top flow detection likelihood. The top flow detection likelihood of a network at time $t$ is defined as the probability that the quantum error is zero: $P(e_{\alpha}(t)=0)$. When the meaning is obvious, we will refer to this value simply as the detection likelihood. §

We can now introduce the detection likelihood equation:

Lemma 6. Detection under partial information. The detection likelihood of a network at time $t$ follows a multivariate hypergeometric distribution as follows:

$$P(e_{\alpha}(t)=0) = P(C_{\alpha}(t) = F_{\alpha}) = \sum_{x' \in Z(t)} \frac{\prod_{i=1}^{\alpha} \binom{x_i(t)}{x_i}}{\prod_{i=1}^{\alpha} \binom{\sum_{j=1}^{n} x_j}{\sum_{j=1}^{n} x_j(t)}}$$

(19)

where $Z(t)$ is the zero quantum error region, expressed as:

$$Z_{\alpha}(t) = \{x' \in \mathbb{N}^{|F|} \mid \sum_{i=1}^{\alpha} x'_i = \sum_{i=1}^{\alpha} x_i(t), \forall x' \leq x \sigma, x'_i > x_j \forall i, j \text{ s.t. } i \leq \alpha, j > \alpha\}$$

(20)

and $a \leq b$ means that $b$ is at least as Pareto efficient as $a$ the Appendix.
Proof. As a test of generality, it is easy to see that equation (19) is a generalization of equation (17) for arbitrary traffic distributions. For a formal proof, see the Appendix. §

3.3. Cutoff Sampling Rates

From a practical standpoint, the detection likelihood $P(\epsilon_a(t))$ cannot be computed for times $t < t_0$ because the size of all flows $\sigma_i$ is only known with certainty at time $t = t_0$. Nevertheless, its equation reveals important properties related to the problem of elephant flow detection. Suppose that a network switch inspects packets in real-time with the goal of timely identifying the top largest flows, where a flow’s size is determined by an arbitrary metric—e.g., by packet counts, byte counts, rate, or most generically, by the $q$-size metric if the QoS utility function $q()$ is known. Assume that, due to limitations in both computing power and memory footprint, the switch can only store in the cache a maximum of $\alpha$ flows. Then, the following statements about the detection likelihood in equation (19) are true:

- It provides the minimum amount of packet samples we need to inspect (equivalently, the minimum amount of time we need to wait) to make a classification decision that will be correct with a probability given by $P(\epsilon_a(t) = 0)$.

- It mathematically quantifies the trade-off between time and the quantum error: if we trade time, we can reduce quantum error; if we trade quantum error, we can make a detection decision sooner.

From an information theory standpoint, a relevant question is to identify the minimum amount of information that needs to be sampled from the traffic dataset in order to detect the largest flows for a given detection likelihood. This problem is similar to the concept of Nyquist rate in the field of signal processing, which identifies the minimum number of samples that need to be taken from a signal in order to fully reconstruct it. We explore this problem in more detail through an example.

Example 1. Minimum sampling rate of well-known heavy tailed traffic distributions. Let $F$ be the set of flows in a network and let $\sigma_i$ be the size of each flow $i$, for $1 \leq i \leq |F|$. Assume that $\sigma_i$ follows any of these well-known distribution functions:

| Distribution | $\sigma_i$ |
|--------------|------------|
| Laplace      | $\sigma_i = 1 - e^{-\beta}$ |
| Cauchy       | $\sigma_i = \frac{1}{\pi(1 + \beta)}$ |
| Sech-squared | $\sigma_i = 1 + e^{-\beta}$ |
| Gaussian     | $\sigma_i = \frac{e^{\beta^2/2}}{\sqrt{2\pi}}$ |
| Linear       | $\sigma_i = \gamma|F| - 1$ |

where $\gamma$ is chosen so that $\sum_i \sigma_i$ is a constant. Figure 5 plots the detection likelihood using equation (19) for the case that $\sum_i \sigma_i = 300$, $\alpha = 5$ and $|F| = 40$ when a fraction $\rho$ of the traffic is sampled, for $0 \leq \rho \leq 1$. The cutoff rates that result in a detection likelihood of 0.99 are also computed. As expected, for non-heavy tailed traffic patterns such as the linear distribution, the cutoff rate is high at $p = 0.97$, while the cutoff rate for heavy tailed patterns such as the Gaussian distribution is much lower at $p = 0.01$. Under the special case where the $q$-size metric corresponds to the number of packets in a flow, this means that for the Gaussian, Laplace, Sech-squared and Cauchy distributions it is enough to sample 1%, 3%, 7%, and 12% of the total traffic dataset, respectively, in order to detect the 5 largest flows with a 99% chances of being correct. §

3.4. High-Performance Detection Algorithms

3.4.1 Base Algorithm: The BubbleCache

A good amount of elephant flow detection algorithms from the literature use packet sampling as a strategy to reduce computational complexity [1, 4, 10, 13, 15]. For instance, Psounis et al. [10] introduce an elegant low-complexity scheduler which relies on packet sampling to detect when a flow traversing a network switch is likely to be an elephant flow. In [1], the idea of packet sampling is generalized to design an actual elephant trap, a data structure that can efficiently retain the elephant flows and evict the mouse flows requiring low memory resources. These existing algorithms, however, treat the packet sampling rate as an input that operators need to manually adjust. The theoretic results described in this paper allow us to develop an unmanned packet sampling algorithm that can dynamically adjust the sampling rate towards tracking a detection likelihood target. We develop this algorithm in this section.

Figure 5. Detection likelihood of some known distributions

We know that heavy tailed traffic characteristics such as those found in real world networks expose detection likelihood curves with well defined cutoff rates, as illustrated in Figure 5. Above the cutoff rate, the gains on the probability to accurately detect the largest flows are small. Below it, the penalties are large. A detection algorithm can benefit from this property by tuning its sampling rate to target the cutoff rate, substantially reducing the computational cost of processing traffic while controlling a small or negligible error rate. This suggests the following simple base algorithm to detect elephant flows at high speed traffic rates:

Pseudocode 1: The base BubbleCache algorithm

8
Algorithm BubbleCache

$C_i$: Targeted accuracy parameter
$\delta_p$: Sampling rate step size
$T_i$: Inactivity timeout
$T_h$: Housekeeping routine timeout
$t$: The current time

Upon receiving a packet from an arbitrary flow $f_i$:

- Sample the packet with a probability $p(t)$.
- If the packet is sampled:
  - If the packet’s flow is not in $C_d(t)$:
    - Add a new flow record to $C_d(t)$ for the packet’s flow;
  - Update $x_i$ according to the $q$-size metric;
- Every $T_h$ units of time:
  - If undersampling(), increase $p(t)$ by $\delta_p$;
  - Otherwise, reduce $p(t)$ by $\delta_p$;
  - Remove flows from $C_d(t)$ that have been inactive for $T_i$;

Function undersampling():

- If $P(e_d(t) = 0)$ is lower than $C_i$, return true;
- Else, return False;

The central idea of the above pseudocode, referred to as the BubbleCache algorithm, is to sample packets at a rate $p(t)$ which is updated to track a target detection likelihood: if the current detection likelihood $P(e_d(t) = 0)$ is lower than a target $C_i$, then increase $p(t)$; otherwise, decrease $p(t)$.

A practical limitation of the BubbleCache algorithm is the calculation of the detection likelihood value, $P(e_d(t) = 0)$, because its formula, introduced in equation (19), requires combinatorial operations that quickly overflow the computational capabilities of modern computers. In the next section we develop a method to overcome this limitation.

3.4.2. Estimating Detection Likelihoods

We introduce the reconstruction lemma which will provide the blueprints of our proposed top flow detection algorithm:

Lemma 7. Reconstruction under partial information. Let $F$ be a set of flows transmitting data over a network and assume that the traffic dataset generated by the flows is heavy tailed according to Definition 13. Let also $x_i$ be the size of flow $f_i$ when traffic is sampled at a rate $p$, for $0 \leq p \leq 1$ and $1 \leq i \leq |F|$. Then the following is true:

(R1) There exists a cutoff sampling rate $p_c$ such that for any sampling rate $p \geq p_c$, $\sigma_i \gg \sigma_j$ implies $x_i \gg x_j$ with high probability.

(R2) The more heavily tailed the traffic data set is (Definition 13) the lower the cutoff sampling rate $p_c$.

(R3) If the sequence $\{x_1, x_2, ..., x_{|F|}\}$ is heavy tailed, then $x_i \gg x_j$ implies $\sigma_i \gg \sigma_j$ with high probability.

(R4) If the sequence $\{x_1, x_2, ..., x_{|F|}\}$ is not heavy tailed, then either $p < p_c$ or the traffic dataset is not heavy tailed, or both.

Proof. See the Appendix. §

The Reconstruction Lemma has practical implications in the design of high performance algorithms to detect elephant flows. In particular, from the Lemma 7/R4, if $\{x_1, x_2, ..., x_{|F|}\}$ is not heavy tailed, then either the traffic has no elephant flows or the sampling rate is too small, $p < p_c$.

Because real world network traffic is heavy tailed (otherwise there would be no need to identify elephant flows to optimize network traffic), we can conclude that $p > p_c$ and hence that the sampling rate needs to be increased. If instead $\{x_1, x_2, ..., x_{|F|}\}$ is heavy tailed, then using Lemma 7/R3 we know that $p > p_c$ and we can decrease the sampling rate. This simple strategy allows the BubbleCache algorithm to operate just in the neighborhood of the frontier defined by $p = p_c$, for some cutoff rate $p_c$ that satisfies the reconstruction lemma.

Now the problem of detecting if a sequence of flow sizes $\{x_1, x_2, ..., x_{|F|}\}$ is heavy tailed is much more tractable than the problem of computing the detection likelihood $P(e_d(t) = 0)$. To this end, we propose to use the fourth standardized moment, known also as the kurtosis [16], which is simple to measure and provides the degree to which a signal is heavy tailed.

Table 1 presents the kurtosis of the traffic data sets introduced in Example 1. As expected, the four heavy tailed data sets (Laplace, Cauchy, Sech-squared and Gaussian distributions) present a high kurtosis (above 12), whereas the non-heavy tailed distribution (linear distribution) exposes a low kurtosis (-1.2). By using the kurtosis measurement, we can know if the sampled traffic dataset is heavy tailed and therefore if the detection likelihood is high according to Lemma 7.

| Distribution | Linear | Laplace | Cauchy | Sech-squared | Gaussian |
|--------------|--------|---------|--------|--------------|----------|
|              | 1.2    | 25.88   | 20.54  | 12.11        | 18.86    |

The next pseudocode provides the adjustment needed on the base algorithm to enable the calculation of the cutoff sampling rate based on the kurtosis method:

Pseudocode 2: undersampling() function with kurtosis

Function undersampling():

- If Kurt[ $\{x_1, x_2, ..., x_{|F|}\}$] is lower than $C_p$:
  - Return true;
- Else:
  - Return False;

4. Performance Benchmarks

We have implemented the BubbleCache as a passive tapping networking device—i.e., a device that processes a mirrored copy of the traffic without affecting any of the active networking equipment (routers, switches, hosts,
The device specifications include two Intel Xeon E5-2670 processors clocked at 2.50 GHz for a total of 20 physical cores with 25.6MB of L3 cache for each processor. It also incorporates four 40 Gbps Solarflare SFC9100 SFP optical interfaces steered by DNAC [11], a high performance packet forwarding engine that performs line rate per-flow load balancing from the network ports to the processor cores. Each core is programmed to run a replica of the BubbleCache algorithm presented in Pseudocode 1 with the undersampling() method based on the kurtosis measurement as described in Pseudocode 2.

We present two sets of benchmarks. First, we measure the performance of the BubbleCache under a controlled lab environment. For these tests, the sampling rate is statically set, which allows us to make fine-grained measurements of the quantum error at various sampling rates. The second set of benchmarks consists of a series of high-performance live tests carried out while running the BubbleCache device during the 2016 SuperComputing (SC) Conference.

Throughout these benchmarks, for lack of a QoS function, we take the number of packets in a flow as the $q$-size metric. (Hence, we assume a metric-based QoS function as in Lemma 1.) If the QoS function is known, the same set of benchmarks can be reproduced using the general definition of the $q$-size metric.

### 4.1. Measurements with Static Sampling

In this section we present the results of testing the BubbleCache device in a controlled lab environment using traffic from our corporation’s local area network. This traffic, which we will refer as the LAN traffic dataset, includes a mix of machine generated flows (for services such as SNMP) and human generated traffic (for applications such as HTTP/HTTPS). The high level statistics of the packet trace are described in Table 2. Because our goal is to measure the performance of the BubbleCache at fixed sampling rates, for the tests in this section we modify Pseudocode 1 to keep the sampling rate constant at a predetermined value of our choice.

#### Table 2. Statistics of the LAN traffic dataset

|        | TCP | UDP | ICMP | Other | Avg pkt size | Size |
|--------|-----|-----|------|-------|--------------|------|
|         | 96.74% | 3.17% | 0.03% | 0.06% | 592.61 | 120GB |
| DHCP   | 0.05% | 2.95% | 4.91% | 0.65% | 49.66% | 41.78% |

#### 4.1.1. Cutoff Sampling Rate

We start our tests by measuring the natural cutoff sampling rate of the LAN traffic. To do this measurement, we replay the trace at the rate it was captured (the LAN traffic dataset had a peak rate of 3474.89 Mbps) to ensure there is no packet loss and measure the quantum error due to sampling by using equation (18). The results are illustrated in Figure 6. We note that the LAN dataset accepts a sampling rate of $p = 0.005$ while maintaining a zero quantum error (QER).

#### Figure 6. Cutoff sampling rate of the LAN dataset

##### 4.1.2. Optimal Sampling Rate at 100 Gbps

In the next experiment, we replay the LAN dataset at a rate of 100 Gbps and measure the QER of the BubbleCache as a function of the sampling rate. The results in Figure 7 illustrate an expected U-shape with three different regions. For very low sampling rates ($p=0.001$ and below), the QER rapidly increases due to excessive sampling. For high sampling rates ($p=0.9$ and above) the QER is also high due to the BubbleCache device not being able to keep up with the 100 Gbps traffic rates, resulting in packet drops. The optimal sampling rate sits somewhere between these two edge cases, at around $p=0.46$ resulting in QER=0.00875.

#### Figure 7. QER at 100 Gbps as a function of sampling

##### 4.1.3. Quantum Error and Packet Drops

Figure 8 presents quantum errors and packet drops at different traffic and sampling rates. The results indicate that using no sampling ($p=1$) is sub-optimal for rates of 77 Gbps and above, in the region where the QER becomes larger than zero (Fig. 8-top). For this region, the QER can be reduced by progressively incrementing the sampling rate to the neighborhood of $p=0.4$ (Fig. 8-middle), in agreement with the results in Figure 7. Increasing the sampling rate
beyond this value (Fig. 8-bottom) helps further reduce packet drops but it has a negative effect on the QER.

4.2. Measurements with Dynamic Sampling

In this section, we test the dynamic version of the BubbleCache algorithm in a live high performance network environment with the goal to: (1) measure the natural cutoff sampling rate of traffic from a real world IP network and its dynamic variations throughout time, (2) measure the convergence and stability of the dynamic sampling rate algorithm and (3) measure the computational and memory footprint savings obtained by operating at the neighborhood of the cutoff sampling rate. These experiments were performed during the days of November 14 through 18, 2016, at the SuperComputing (SC) venue as part of the high performance computing (HPC) demonstrations run in the SCinet network. The SCinet HPC network is built every year to help support the SC venue and to test new technologies in a realistic network environment. This large scale network environment supports a traffic mix of both small flows generated by thousands of users on the conference floor and very large flows generated by large scale, big data science experiments carried out from the booths, resembling the traffic conditions typically found in Research and Education networks such as ESnet and Internet2 [12].

As part of the SC/SCinet team, we connected the 4x40 Gbps ports of the BubbleCache device to one of the network taps which had full visibility of the SC/SCinet traffic. For these tests, the BubbleCache algorithm was configured with the following parameters: $C_t = 100$ (target kurtosis value), $\delta_p = 0.01$ (sampling rate step size), $T_i = 20$ seconds (connection inactivity timeout), $T_h = 0.05$ seconds (housekeeping routine timeout). The rationale for choosing a target kurtosis value of 100 is to conservatively operate the algorithm at a region where the quantum error is zero with very high probability. Notice that heavy tailed functions such as those presented in Examples 5 and 6 (Laplace, Cauchy, Sech-squared and Gaussian distributions) have kurtosis values between 10 and 25; hence, a value of 100 ensures that the sampled traffic dataset is very heavy tailed. From Lemma 7, this in turn implies that the algorithm operates at the zero quantum error region with high probability. To test the efficacy of the BubbleCache device under limited computing resources, we configured the device to only use four cores out of its total 20 cores, with each core processing one of the four 40 Gbps network ports, and leaving the other 16 cores idle.

4.2.1. Cutoff Sampling Rate for IP Traffic and Convergence Measurements

While the existence of cutoff sampling rates was mathematically shown in Lemma 7 (Figure 5), a question of interest is whether their presence can also be detected on real live traffic. In particular, we are interested in answering: what is the cutoff sampling rate of a real world IP network? and how does this cutoff rate change as traffic patterns in the network change throughout the day?

Figure 9 presents the sampling rate obtained from running the BubbleCache algorithm for traffic generated from the SC/SCinet network during high and low traffic hours. The traffic rate coming from the venue floor during high hours (during the day) was around 25 Gbps with peaks at 60 Gbps, whereas at low traffic hours (at night) traffic was around 1Gbps or below. With a target kurtosis of 100, the
cutoff sampling rate at high and low traffic hours is around 0.001 and 0.01, respectively. This result shows that at traffic rates of about 25 Gbps, we can sample around 1 out of 1000 packets (a computational cost reduction of 1000 times) and still capture all the largest flows with high probability as the resulting sampled traffic dataset is very heavy tailed. Another result worth noticing is that the higher the traffic rates, the lower we can reduce the sampling rate for a fixed target kurtosis level (i.e., a fixed degree of heavy tailedness). Using Lemma 7/R2, this implies that network traffic is more heavy tailed during the day, which is in agreement with the fact that more heavy tailed traffic is produced during the conference hours when the big data science experiments launched throughout the day are combined with thousands of user-generated small flows. This result is relevant in that it is at very high speed rates that a reduction of the sampling rate becomes most valuable from an algorithmic scalability point of view. The BubbleCache algorithm is able to leverage this natural property of the traffic by tuning the sampling rate up or down as necessary.

Figure 10 plots the convergence of both the sampling rate and the kurtosis measurement as the algorithm is started from two different initial conditions during the high traffic hours (around 2:10pm). In Figure 10-top, the initial sampling rate is set to 0.0001, ten times below the optimal rate of 0.001, while in Figure 10-bottom, the initial sampling rate is set to 0.01, ten times above it. In both cases, in a few seconds the algorithm converges to the same cutoff sampling rate around 0.001. The convergence time is linear with time and its slope can be tuned by adjusting the sampling rate step size \( \delta_p \) and the housekeeping routine timeout \( T_h \). While left outside the scope of these results, an area of optimization is to improve the convergence time by using an adaptive heuristic that increases the step size if the kurtosis index is far from the target and reduces the step size as it gets closer to it.

**Figure 9. Measurements of the cutoff sampling rate**

**Figure 10. Convergence of the BubbleCache algorithm**

In summary, the above plots show that, regardless of the initial conditions, the sampling rate converges to the targeted kurtosis value of 100 and, upon convergence, both the sampling rate and the kurtosis parameters stay stable around their targets.

4.2.2. Memory Footprint

In addition to the computational savings shown in the previous section, sampling also has a positive effect on the memory footprint requirements of the algorithm: the higher the sampling rate, the smaller the size of the flow cache as more flows are filtered out. We are now interested in measuring the memory footprint reduction accomplished as a consequence of sampling traffic at the targeted kurtosis level.

Figure 11 illustrates the size of the BubbleCache as a function of time as the algorithm converges to the cutoff rate of 0.001 from an initial sampling rate of 0.01. (This plot captures the same time period as the plot in Figure 10-bottom.) The total number of active flows in the SC/Scinet network for this period is around 25,000. As the BubbleCache algorithm is initiated, since the sampling rate is substantially above the cutoff rate, the size of the flow cache steadily increases reaching more than 2000 flow entries. Then as the sampling rate and the kurtosis level continue to decrease, the size of the cache begins to decrease until it reaches a stable point once the targeted kurtosis level of 100 is achieved. In steady state and with 25,000 active flows, the size of the flow cache stabilizes around 250 flows, which represents a 100 time reduction in memory size.

**Figure 11. Size of the BubbleCache**
5. Conclusions and Forthcoming Work

The theory of flow ordering described in this paper leads to a variety of applications that can be integrated into modern computer networks. In the short term, our work currently focuses on integrating the BubbleCache algorithm as part of a commercial software defined network (SDN) data plane to operate at port rates of 100 Gbps. We are packaging the BubbleCache in two formats: (1) as a top flow detection and ordering algorithm for SDN networks using sFlow [6] and (2) as a high performance queue for the real time separation of elephant and mouse flows to isolate and protect them from each other.

Beyond the problem of elephant flow detection, the theory of flow ordering has a broader set of applications in the context of network diagnostics and optimization. One interesting use case is the construction of a top command for communication networks. In the world of computers, the top command found in UNIX like terminals displays in real time the processes that have the largest performance impact on the system resources (CPU utilization, memory, I/O, etc.). In the context of computer networks, a real time top tool providing the largest flows ordered according to their QoS impact would bring value at least in two areas: (1) as a diagnostic tool, to help network operators quickly spot those flows incurring the largest impact to the overall QoS of the network, potentially signaling network misconfiguration such as bogus BGP routes; (2) as part of an automated closed loop traffic engineering module (such as those made possible in SDN networks) to automatically detect large flows and reroute them towards improving the network QoS. In the mid and longer term, our plan is to use the theory of flow ordering introduced in this paper to address this broader set of network diagnostics and optimization problems.

References

[1] Y. Lu, M. Wang, B. Prabhakar, F. Bonomi, “ElephantTraps: A low cost device for identifying large flows,” IEEE Symposium on High-Performance Interconnects, 2007.
[2] T. Fioreze, L. Granville, R. Sadre, and A. Pras, “A Statistical Analysis of Network Parameters for the Self-management of Lambda-Connections,” Scalability of Networks and Services Volume 5637 of the series Lecture Notes in Computer Science, pp 15-27.
[3] K. Lan, J. Heidemann, “A measurement study of correlations of Internet flow characteristics,” Elsevier Science, February 2006.
[4] Y. Zhang, B. Fang, Y. Shang, “Identifying High-Rate Flows Based on Bayesian Single Sampling,” International Conference on Computer Engineering and Technology, 2010.
[5] S. Sarvotham, R. Riedi, R. Baraniuk, “Connection-level Analysis and Modeling of Network Traffic,” ACM SIGCOMM Internet Measurement Workshop 2001.
[6] P. Phaal, S. Panchen, N. McKee, “InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks,” RFC 3176, IETF, September 2001.
[7] T. H. Cormen, C. L. Leiserson, R. Rivest, C. Stein, “Introduction to Algorithms,” MIT Press, August 2009.
[8] Dan Gusfield, “Partition-distance: A problem and class of perfect graphs arising in clustering,” Information Processing Letters Volume 82, Issue 3, 16 May 2002, pp 159-164.
[9] W. Willinger, V. Paxson, and M.S. Taqqu, “Self-similarity and Heavy Tails: Structural Modeling of Network Traffic,” in A Practical Guide to Heavy Tails: Statistical Techniques and Applications, Adler, R., Feldman, R., and Taqqu, M.S., editors, Birkhauser, 1998.
[10] K. Psounis, A. Ghosh, B. Prabhakar, “SIFT: A simple algorithm for tracking elephant flows, and taking advantage of power laws,” Conference on Control, Communication and Computing, 2005.
[11] Jordi Ros-Giralt, Alan Commike, Daniel Honey, Richard Lethin, “High-Performance Many-Core Networking: Design and Implementation,” In ACM Innovating the Network for Data-Intensive Science (INDIS), Austin, TX, USA, ACM, November, 2015.
[12] Research and Education Network [Online]: https://en.wikipedia.org/wiki/National_research_and_education_network
[13] C. Estan, K. Keys, D. Moore, and G. Varghese, “Building a better netflow,” ACM SIGCOMM, 2004.
[14] Kun-Chan Lan, John Heidemann, “A measurement study of correlations of Internet flow characteristics,” Elsevier Science, 2006.
[15] T. Mori, M. Uchida, R. Kawahara, “Identifying elephant flows through periodically sampled packets,” Internet Measurement Conference, 2004.
[16] Philippe Pebay, “Formulas for Robust, One-Pass Parallel Computation of Covariances and Arbitrary-Order Statistical Moments,” Sandia National Labs Report SAND2008-6212, September 2008.
[17] Willinger, W. ; AT&T Bell Labs., Murray Hill, NJ, USA ; Taqqu, M.S. ; Sherman, R. ; Wilson, D.V., “Self-similarity through high-variability: statistical analysis of Ethernet LAN traffic at the source level,” IEEE/ACM Transactions on Networking, Feb 1997.
[18] Z. Yan, M. Veeraraghavan, C. Tracy, C. Guok, “On How to Provision Quality of Service (QoS) for Large Dataset Transfers,” The Sixth International Conference on Communication Theory, Reliability, and Quality of Service (CTRQ 2013), April 21 - 26, 2013.
Appendix. Mathematical proofs.

**Lemma 2. General complexity.** The set $I(QSD)$ includes P, NP and NP hard problems. That is, $I(QSD) \subseteq NP$ and $I(QSD) \cap P \neq \emptyset$.

**Proof.** First we notice that any problem in $I(QSD)$ is verifiable in polynomial time and hence NP, since we can check any arbitrary QoS partition $< F_1, F_2, ..., F_i >$ by just computing the value of $q(F_1, F_2, ..., F_i)$. We will demonstrate that it is possible to construct problems in $I(QSD)$ that are P and NP hard.

Let $< N, F, P_1, P_2, q() >$ be in $I(QSD)$ and assume that $q()$ is defined as in equation (5). From Lemma 1, we know that a solution to the problem instance $< N, F, P_1, P_2, q() >$ can be found by constructing the ordered set $F_e = \{ f_1, f_2, ..., f_{l_e} \}$ according to the rule $m(f_1) > m(f_i)$, and choosing the smallest value of $\lambda^*$ such that $\sum_{v \in \lambda^*} r(f_i) \leq C$. The construction of $F_e$ can be done in $O(|F| \cdot \log(|F|))$ and $\lambda^*$ can be found in $O(|F|)$. Hence the problem instance $< N, F, P_1, P_2, q() >$ belongs to P.

Now let $< N, F, P_1, P_2, ..., P_n, q() >$ be in $I(QSD)$ and assume that $q()$ is defined as follows:

$$q(F_1, F_2, ..., F_i) = \text{rand}(s)$$

where $\text{rand}(s)$ is a random real number generated from a given seed $s$. Because the values of $q(F_1, F_2, ..., F_i)$ are randomly chosen at the time the problem is created from the given seed, we can only resolve $< N, F, P_1, P_2, ..., P_n, q() >$ by trying out all possible QoS partitions $< F_1, F_2, ..., F_i >$. Since from equation (2) we have $q: [F] \Rightarrow \mathbb{R}$, there are $|F|$ possible problem solutions, hence the problem instance $< N, F, P_1, P_2, ..., P_n, q() >$ is NP hard. §

**Lemma 3. Decreasing returns to gains for metric-based ordering.** All metric-based partitioned QoS functions (see Lemma 1) have decreasing returns to gains. Further, for these class of problems, the size of flow $f_j$ is $\sigma(f_j) = \pi(f_j)$ if it is an elephant flow and $\sigma(f_j) = \rho(f_j)$ if it is a mouse flow.

**Proof.** Consider the metric-based partitioned QoS problem described in Lemma 1. Using the condition $m(f_i) > m(f_{i+1})$ and equation (6), we can apply equation (8) to derive the flow size ordering as follows:

$$q(\emptyset, F) = C_0 - \sum_{f \in F} \pi(f) = q_0$$

$$q(F_1, F \setminus F_1) = q_0 + \pi(f_1) = q_1$$

$$q(F_2, F \setminus F_2) = q_1 + \pi(f_2) = q_2$$

$$\vdots$$

$$q(F_{\lambda^*}, F \setminus F_{\lambda^*}) = q_{\lambda^*} + \pi(f_{\lambda^*}) = q_{\lambda^*}$$

$$q(F_{\lambda^*+1}, F \setminus F_{\lambda^*+1}) = q_{\lambda^*+1} + \pi(f_{\lambda^*+1}) = q_{\lambda^*+1}$$

$$\vdots$$

$$q(F_{l-1}, F \setminus F_{l-1}) = q_{l-1} - \pi(f_{l-1}) = q_{l-1}$$

$$q(F, q) = q_l - \pi(f_l) = q_l$$

where $F_i = F_{l-1} \cup \{ f_i \}$, $1 \leq i \leq l$, $F_0 = \emptyset$ and $F_l = F$.

**Decreasing returns to gains for metric-based QoS.**

The above sequence has two separated segments. The first segment generates the sequence $\{ \emptyset, F_1, F_2, ..., F_{\lambda^*} \}$ which corresponds to all possible $\lambda^*$-optimal partitions under the assumption that the network is congested—upper level of equation (5). Under this region, moving a flow $f_i$ from policy $P_2$ to policy $P_1$ increases the QoS of the network by $\pi(f_i)$—i.e., the QoS penalty generated by flow $f_i$ when scheduled as high priority. The second segment generates the sequence $\{ F_{\lambda^*}, F_{\lambda^*+1}, ..., F_l \}$ which corresponds to all possible $\lambda^*$-optimal partitions under the assumption that the network is not congested—lower level of equation (5). Under this region, moving a flow $f_i$ from policy $P_2$ to policy $P_1$ decreases the QoS of the network by $\rho(f_i)$—i.e., the QoS gain foregone by flow $f_i$ when scheduled as low priority. The optimal partition is reached when $\lambda = \lambda^*$, which is accomplished with the configuration $< F_{\lambda^*}, F \setminus F_{\lambda^*} >$ and the total QoS is $C_0 - \sum_{f \in F_{\lambda^*}} \rho(f)$ with decreasing returns to gain. §

**Lemma 4. Nested convexity and inclusiveness.** A 2-partitioned QoS function is inclusive if and only if it is nested convex in the vicinity of all its $\lambda^*$-optimal partitions.
Proof. Let $< N, F, P_1, P_2, q() >$ be a 2-partitioned QoS problem and assume that $q()$ is nested convex in the vicinity of its $\lambda$-optimal partitions but not inclusive. Then, from Property 1, there must be a $\lambda$-optimal partition which does not include the $(\lambda - 1)$-optimal partition, for some $\lambda$ between 2 and $|F|$. For such $\lambda$, we have that $P_{\lambda - 1} \neq P_{\lambda - 1} \land P_{\lambda}$ and, from equation (12), the following holds:

$$p_{\lambda} \notin n_0(P_{\lambda - 1})$$

(14)

Since $p_{\lambda}$ is a $\lambda$-optimal partition, we also have that:

$$q(p_{\lambda}) > q(p') \text{ for all } p' \in n_0(p_{\lambda - 1})$$

(15)

But equations (14) and (15) contradict the nested convex definition in equation (13). This proves the “if” part of the lemma.

Assume now that $q()$ is inclusive but not nested convex in the vicinity of some $\lambda$-optimal partition $p_{\lambda}$ and let $p_{\lambda + 1}$ be a $(\lambda + 1)$-optimal partition. By definition, we have that $p_{\lambda + 1}$ is in the partition neighborhood of $p_{\lambda}$, that is, $p_{\lambda + 1} \in n_0(p_{\lambda})$.

From equation (18) and since $q()$ is not nested convex in the vicinity of $p$, we also have that $q(p_{\lambda + 1}) > q(p')$ for all $p' \in n_0(p_{\lambda})$. This implies that $p_{\lambda + 1}$ is not in the nested neighborhood of $p_{\lambda}$, and hence that $P_{\lambda} \neq P_{\lambda} \land P_{\lambda + 1}$, which contradicts the assumption that $q()$ is inclusive from Property 1.

Lemma 5. Computational complexity under nested convexity. Let $< N, F, P_1, P_2, q() >$ be a 2-partitioned QoS function that is nested convex in the vicinity of all its $\lambda$-optimal partitions. Then $< N, F, P_1, P_2, q() >$ belongs to $P$.

Proof. Let $< N, F, P_1, P_2, q() >$ be inclusive. Then we can construct the sequence of flows $\{f_1, f_2, ..., f_{|F|}\}$ following equation (8):

$$q(F_1, F \setminus F_1) \geq q(F', F \setminus F')$$

where:

$$F_i = F_{i-1} \cup \{f_i\}, f_i \notin F_{i-1},$$

$$F' = F_{i-1} \cup \{f'\} \text{ for all } f' \notin F_{i-1} \text{ and } F_0 = \{\emptyset\}$$

Each step in the above recursive equation requires the verification of at most $|F|$ possible solutions. Since the above recursivity terminates in $|F|$ iterations, the total cost of finding the sequence $\{f_1, f_2, ..., f_{|F|}\}$ is $O(|F|^2)$. Once we have the ordered list of flows, we can find the optimal solution by identifying the value of $\lambda'$ that maximizes $q(F_c, F \setminus F_c)$, where $F_c = \{f_1, f_2, ..., f_{\lambda'}\}$. The cost of finding $\lambda'$ is at most $O(|F|)$. Hence, all 2-partitioned problems that are inclusive can be solved in $O(|F|^3)$ and belong to $P$.

Lemma 6. Detection under partial information. The detection likelihood of a network at time $t$ follows a multivariate hypergeometric distribution as follows:

$$P(e_{\alpha}(t) = 0) = P(C_{\alpha}(t) = F_{\alpha}) = \sum_{\forall x \in Z(t)} \frac{\prod_{i=1}^{\sigma_i} \binom{a_i}{x_i}}{\sum_{\forall x} \binom{a_i}{x_i}}$$

(19)

where $Z(t)$ is the zero quantum error region, expressed as:

$$Z_{\alpha}(t) = \{x' \in \mathbb{N}^{|F|} \mid \sum_{\forall i} x'_i = \sum_{\forall i} x_i(t), x'_i \leq \sigma, x'_i > x_i(t) \forall i, j \text{ s.t. } i \leq \alpha, j > \alpha\}$$

(20)

and $a \leq_p b$ means that $b$ is at least as Pareto efficient as $a$.

Proof. Assume a discrete fluid model of the network in which each flow $i$ needs to transmit a number of water droplets equal to its $q$-size metric $\sigma_i$. Flows transmit water through the network one droplet at a time and each droplet is transmitted at arbitrary times. By convention, we will assume that the first and last droplets from any of the flows are transmitted at times $0$ and $t_\ast$, respectively. An observer of the network performs only one task: counting the number of droplets each flow has transmitted and storing such information in a vector $x_i(t)$, where each component $x_i(t)$ corresponds to the amount of droplets seen from flow $i$ up until time $t$. Based on this information, the objective is to quantify the probability that the set of flows $C_{\alpha}(t)$ is the same as the set of flows in $F_{\alpha}$.

At time $t$, the total number of droplets transmitted is $\sum_{\forall i} x_i(t)$ out of a total number of $\sum_{\forall i} \sigma_i$ droplets. The total number of possible ways in which $\sum_{\forall i} x_i(t)$ droplets are transmitted is given by this expression:

$$\left( \sum_{\forall i} \sigma_i \right) \left( \sum_{\forall i} x_i(t) \right)$$

(21)

Only a subset of the total number of ways in which droplets are transmitted correspond to the case of zero quantum error. In particular, those vectors $x'$ that satisfy the following condition:

- The total number of droplets transmitted, $\sum_{\forall i} x'_i(t)$, is equal to $\sum_{\forall i} x_i(t)$.
- The number of droplets transmitted by a flow cannot be larger than its $q$-size metric: $x'_i \leq \sigma_i$. (Where $a \leq_p b$ means that $b$ is at least as Pareto efficient as $a$.)
- The top $\alpha$ flows, $f_1, f_2, ..., f_\alpha$, are captured by the set $C_{\alpha}(t)$, that is, $x'_i > x'_j$ for all $i$ and $j$ such that $i \leq \alpha$ and $j > \alpha$. 

The above three conditions define the zero quantum error region, \( Z_0(t) \), and the total number of ways in which droplets can be transmitted to generate a vector \( x' \) that belongs to \( Z_0(t) \) is as follows:

\[
|Z_0(t)| = \sum_{x \in Z_0(t)} \prod_i \sigma_i^{x_i}(x_i')
\]  

(22)

The probability that the quantum error is zero, \( P(e_0(t) = 0) \), can now be obtained from the division of equation (22) by equation (21). §

**Lemma 7. Reconstruction under partial information.** Let \( F \) be a set of flows transmitting data over a network and let \( x_i \) be the size of flow \( f_i \) when traffic is sampled at a rate \( p \) for \( 0 \leq p \leq 1 \). Then there exists a sampling rate \( p_c \) such that:
- If \( x_i \gg x_j \) and \( p \geq p_c \), then \( \sigma_i \gg \sigma_j \) with high probability.
- If \( \sigma_i \gg \sigma_j \) and \( p \geq p_c \), then \( x_i \gg x_j \) with high probability.

**Proof.** Consider the first statement and suppose that there exists a pair of flows \( f_i \) and \( f_j \) such that \( x_i \gg x_j \). There are three possible cases:

\[
\begin{align*}
\sigma_i &> \sigma_j \\
\sigma_i &= \sigma_j \\
\sigma_i &< \sigma_j.
\end{align*}
\]

Assume that \( \sigma_i = \sigma_j = \sigma \) and let \( X_k \) be a random variable such that \( X_k = x_k \) when traffic is sampled at a rate \( p \). Since \( \sigma_i = \sigma_j \), then by symmetry the expected value of both \( X_i \) and \( X_j \) must be the same and equal to \( \bar{x} = (x_i + x_j)/2 \). (Notice that the total number of samples taken for flows \( f_i \) and \( f_j \) must stay constant and be equal to \( x_i + x_j \).) Let \( p(\rho) \) be the probability of the event \( X_i = x_i \cap X_j = x_j \cap x_i \gg x_j \) divided by the probability of the expected event \( X_i = \bar{x} \cap X_j = \bar{x} \) when traffic is sampled at a rate \( p \). Since \( X_i = \bar{x} \cap X_j = \bar{x} \) is the expected outcome when \( \sigma_i = \sigma_j \), this parameter provides a measurement of the likelihood that \( \sigma_i = \sigma_j \) is true when \( x_i \gg x_j \); if \( p(\rho) \) is close to zero, then \( x_i \gg x_j \) is much less likely than the expected outcome, making the assumption \( \sigma_i = \sigma_j \) unlikely; if \( p(\rho) \) is close to 1, then \( x_i \gg x_j \) is as likely as the expected outcome, which makes \( \sigma_i = \sigma_j \) possible. We have that:

\[
\rho(\rho) = \frac{P(X_i = x_i \cap X_j = x_j \cap x_i \gg x_j)}{P(X_i = \bar{x} \cap X_j = \bar{x})} = \frac{\binom{\sigma_i}{x_i}\binom{\sigma_j}{x_j}}{\binom{\sigma_i + \sigma_j}{\bar{x}}} = \frac{\sigma_i!}{x_i!} \frac{\sigma_j!}{x_j!} \frac{(\sigma - x_i)!}{\sigma_i!} \frac{(\sigma - x_j)!}{\sigma_j!} = \frac{\bar{x}!}{x_i!} \frac{(\sigma - \bar{x})!}{(\sigma - x_j)!} \frac{(\sigma - x_j)!}{x_j!} \frac{(\sigma - x_i)!}{x_i!} \frac{\bar{x}!}{(\sigma - \bar{x})!} \frac{(\sigma - \bar{x})!}{\sigma_i!} \frac{(\sigma - \bar{x})!}{\sigma_j!} = \frac{\bar{x} \cdot \cdots \cdot (x_j + 1)}{(\sigma - \bar{x}) \cdot \cdots \cdot (\sigma - x_i + 1)} \frac{\rho(p)}{\rho(p)^{|F|}} \leq 1
\]

(23)

The following must be true:

- Since \( x_i \gg x_j \), then \( P(X_i = x_i \cap X_j = x_j \cap x_i \gg x_j) \) is smaller than \( P(X_i = \bar{x} \cap X_j = \bar{x}) \), for \( 0 \leq p \leq 1 \). Hence,

\[
\rho(p) < 1 \quad \text{for} \quad 0 < p < 1
\]

(24)

- If \( p \to 0 \), then \( x_i \to 0 \) and \( x_j \to 0 \), which means that \( P(X_i = x_i \cap X_j = x_j \cap x_i \gg x_j) \to P(X_i = \bar{x} \cap X_j = \bar{x}) \):

\[
\lim_{p \to 0} \rho(p) = 1
\]

(25)

- If \( p \to 1 \), then \( x_i \to \sigma_i = \sigma \) and \( x_j \to \sigma_j = \sigma \), which means that \( P(X_i = x_i \cap X_j = x_j \cap x_i \gg x_j) \to 0 \) and \( P(X_i = \bar{x} \cap X_j = \bar{x}) \to 1 \):

\[
\lim_{p \to 1} \rho(p) = 0
\]

(26)

That is: \( \sigma_i = \sigma_j \) is false when \( p = 1 \), unlikely when \( p \to 1 \), and possible when \( p \to 0 \). Because \( \rho(p) \) is continuous with \( p \), there must exist a \( p_c \) such that if \( p > p_c \), then \( \sigma_i = \sigma_j \) is unlikely.

Now consider the case \( \sigma_i < \sigma_j \). Since \( x_i \gg x_j \), then \( \sigma_i < \sigma_j \) is less likely than \( \sigma_i = \sigma_j \), which means that equations (24), (25) and (26) are also true. Hence we must conclude that if \( x_i \gg x_j \) and \( p > p_c \), then \( \sigma_i = \sigma_j \) with high probability.

Finally, from equation (23), the value of \( \rho(p) \) rapidly decreases as the value \( x_i - x_j \) increases. This means that the higher the value \( x_i - x_j \), the more likely the value \( \sigma_i - \sigma_j \) is higher too, provided that \( p > p_c \). Hence, \( x_i \gg x_j \) and \( p \geq p_c \) imply \( \sigma_i \gg \sigma_j \) with high probability, which proves the first statement in the lemma.

The second statement in the lemma can also be demonstrated following a similar approach and is omitted from this text. §

**Corollary from Lemma 7. Reconstruction properties under partial information.** Let \( F \) be a set of flows transmitting data over a network and assume that the traffic dataset generated by the flows is heavy tailed according to Definition 13. Let also \( x_i \) be the size of flow \( f_i \) when traffic is sampled at a rate \( p \), for \( 0 \leq p \leq 1 \) and \( 1 \leq i \leq |F| \). Then the following is true:
There exists a cutoff sampling rate $p_c$ such that for any sampling rate $p \geq p_c$, $\sigma_i \gg \sigma_j$ implies $x_i \gg x_j$ with high probability.

The more heavy tailed the traffic data set is (Definition 13) the lower the cutoff sampling rate $p_c$.

If the sequence $\{x_1, x_2, ..., x_{|F|}\}$ is heavy tailed, then $x_i \gg x_j$ implies $\sigma_i \gg \sigma_j$ with high probability.

If the sequence $\{x_1, x_2, ..., x_{|F|}\}$ is not heavy tailed, then either $p < p_c$ or the traffic dataset is not heavy tailed, or both.

Proof: R1 is a restatement of Lemma 7 applied to the case of heavy tailed traffic. R2 can be easily seen from equation (23) and Definition 13: the more heavy tailed a traffic data set is, the larger the value of $x_i - x_j$; as a result, $p(p)$ becomes smaller, which means that the cutoff sampling rate also becomes smaller. R3 is also a restatement of Lemma 7. R4 is true because it is the negative form of Lemma 7: if $p \geq p_c$ and the traffic dataset is heavy tailed, then we know from Lemma 7 that $\sigma_i \gg \sigma_j$ implies $x_i \gg x_j$ with high probability; but that contradicts the assumption that $\{x_1, x_2, ..., x_{|F|}\}$ is not heavy tailed. §