A Scalable Monitoring Approach Based on Aggregation and Refinement

Yow-Jian Lin, *Member, IEEE* and Mun Choon Chan, *Member, IEEE*

***Abstract*—Network monitoring is an integral part of any network management system. In order to ensure end-to-end service quality stated in service level agreements (SLAs), managers of a service provider network need to gather quality-of-service (QoS) measurements from multiple nodes in the network. For a large network with over thousands of flows with end-to-end SLAs, the information exchanged between network nodes and a central network management system (NMS) could be substantial.**

In this work, we propose a mechanism called aggregation and refinement based monitoring (ARM) to reduce the amount of information exchange. ARM is a generic mechanism that can be configured to run with different objectives, including threshold-based, rank-based and percentile-based. The mechanism enables the NMS to collect data from network nodes using a dynamic QoS data aggregation/refinement technique, and to process these information differently depending on its measurement objective.

Our simulation results show that for these various objectives, the selective refinement process is able to validate SLAs quickly, is an order of magnitude more efficient than a simple polling scheme, and performs well across a wide range of traffic loads.

***Index Terms*—Aggregation, monitoring, network management, refinement, service level agreement, validation.**

I. Introduction

THE MONITORING of end-to-end quality-of-service (QoS) is increasingly critical to Internet service providers (ISPs). QoS guarantee has become a highly desirable feature in Internet service offering. An ISP must collect QoS statistics through monitoring to convince its customers that it has met the QoS guarantees stated in respective service level agreements (SLAs). It is also important for a service provider to constantly monitor network status in order to detect/predict QoS violation and to drive network control.

The needs of constant QoS monitoring is even more apparent in measurement-based approaches to resource provisioning [7]. Traditionally, ISPs have been over-provisioning resources to meet their service level agreements with customers, an approach that is not cost effective. Recent works on resource allocation [10] and [6] that build on both deterministic and statistical models have yielded interesting results. Nevertheless, the provision based on these results is still conservative. On the other hand, a measurement-based approach starts from a roughly-estimated provisioning. It then adapts to changes in resource needs based on constant interactions between measurement and provisioning adjustment. A timely, efficient QoS monitoring is, thus, the key to a successful measurement-based approach to ensuring QoS offering.

One of the challenges in monitoring is the collection of measurement data, in particular, when managing a large network with many customer flows. The kind of flows of interest in this paper, called SLA flows, is between any two end points in an ISP network, and is an aggregated traffic governed by an SLA. SLA flows are long lasting; once an SLA flow is admitted, the flow usually stays up for an extended period of time. For an ISP managing a network consisting of a large number (>100) of network devices and a large number (>1000) of flows per device, the amount of information collected and processed can be substantial.

This paper describes a scalable and efficient framework for a central network management system (NMS) of an ISP to collect QoS measurement data from network devices for SLA validation. In accordance with network management terminologies, we refer to the object that collects and sends measurement data at each router as an *agent*. We also use the terms NMS and *manager* interchangeably. Each agent collects QoS data of SLA flows on a per-hop basis, and the NMS is responsible for assembling the per-hop data it receives from agents to determine the end-to-end QoS of each flow. In this paper, the QoS parameters of interest are *end-to-end packet loss* and *queuing delay*.

The proposed monitoring approach, called aggregation and refinement based monitoring **ARM**, deals with three different monitoring objectives. With a *threshold-based* objective, all flows with QoS parameters exceeding (or below) some thresholds X are to be identified. For example, the manager may want to identify all SLA flows with end-to-end packet loss exceeding 1%. With a *rank-based* objective, the top Nth flows with respect to some QoS parameters are to be identified. For example, the manager may want to identify the ten SLA flows with the highest end-to-end packet loss. Finally, with a *percentile-based* objective, the Nth percentile of a QoS parameter is to be identified. For example, the manager may want to determine the 98th percentile of end-to-end queuing delay of all SLA flows. We believe that these three objectives provide answers to an important subset of the questions asked by a NMS.

Data aggregation is a technique to control the overhead of data exchange. By aggregation we mean that each agent first partitions the set of flows it governs into a small number of groups, then uses a value range (*minimum* and *maximum* values) to approximate the QoS values of the flows in each group. In order for the manager to properly extract information

Manuscript received March 8, 2001; revised December 27, 2001.

The authors are with Networking Research Laboratory, Bell Laboratories, Lucent Technologies, Holmdel, NJ 07733 USA (e-mail: yjlin@research.bell-labs.com; munchoon@research.bell-labs.com).

Publisher Item Identifier S 0733-8716(02)04707-8.

from aggregated data, it must figure out, for each value range, the corresponding group of flows that the range is associated with. To group flows as dynamically as possible (to generate a close approximation) and yet not to explicitly identify flows in each group (to minimize data exchange overhead) is a main challenge.

The design of the **ARM** framework is based on the following two observations: implicit conveying of group membership through shared configuration; and selective refinement based on monitoring objectives.

First, the manager has the route for each individual SLA flow since these flows are typically traffic engineered based on technology such as multiprotocol label switching (MPLS) [13]. As a result, the manager also knows all the SLA flows each agent is monitoring. With these shared configuration, and using the same ordering scheme, such as the lexicographical order of flow identifiers, the manager and agents can refer to a group of flows by using the index and length from the list of sorted SLA flows instead of enumerating the flow identifiers of interest. This observation motivates the proposed data aggregation scheme presented in Section III-D.

The second observation is that, in order to validate many QoS guarantees, the manager only needs fairly good QoS estimates from a small number of flows. This observation motivates the objective-dependent selective refinement strategies. Details are presented in Section IV.

Note that the reduction of data exchange overhead between NMS and agents comes at the cost of additional aggregation computation by agents at network devices. Given that modern routers are beginning to provide hardware-assisted packet accounting and have large processing capabilities, this appears to be a reasonable tradeoff.

We conducted extensive simulations to study the performance of **ARM** in terms of monitoring overhead reduction. In particular, we studied its performance using different monitoring objectives and under various network load, and aggregation granularity.

The monitoring algorithm proposed here is independent of other SLA management mechanisms, such as admission control and bandwidth/buffer allocation schemes. The aggregation and refinement are also independent of the QoS parameter being monitored; the NMS maintains its responsibility for interpreting the data end-to-end.

This document is organized as follows. Section II discusses related work. Section III presents our monitoring framework, **ARM**, followed by the description of objective-dependent refinement strategies in Section IV. Section V explains the simulation setup and results. Concluding remarks are in Section VI.

II. Related Work

Much effort on network monitoring has been devoted to provide a unified monitoring framework including common protocols for fetching management information, syntax for defining monitoring information and management information. The most popular protocols for network monitoring are the IETF simple network management protocol (SNMP) [1], [2] and the ISO common management information protocol (CMIP). Many management information bases (MIBs) have been defined, including the remote network monitoring management information base (RMON MIB) [14], [15]. RMON provides significant expansion in SNMP functionality, including support for off-line operations, more sophisticated data processing and multiple managers. A drawback with these MIBs is that the MIB data tend to be fairly low level and focus on counters for hardware statistics and errors. A recent development is the definition of a MIB module for performance management of service level agreements (SLA) [16].

In a large network where the amount of management information available is enormous, the collection and processing of these information become the bottleneck. A common approach to reducing monitoring overhead is to vary the polling frequencies based on the state and characteristics of variables being monitored. References [5], [8], and [17] present different approaches to how the polling frequencies can be varied. However, no data aggregation is performed, which limits the overhead reduction achievable.

In [9], the amount of information to be collected is reduced by only collecting information that is required to satisfy the objective of monitoring. For example, if the end-to-end delay of a specific path is required, then only performance data of delay along the specific path will be collected. An inference engine is used to map a request to the individual measurement components. In [3], an instantiation of **ARM** using threshold-based objective is presented and evaluated. This paper presents a much more powerful **ARM** framework with a generic algorithm that can be applied to multiple QoS monitoring objectives.

End-to-end measurements per SLA flow is ideal for deciding if a flow meets its SLA. A large scale end-to-end measurement of packet dynamics over the Internet can be found in [11]. A discussion of using operation and management (OAM) cells to measure end-to-end performance over a ATM network can be found in [4]. While such measurements are appropriate for determining the end-to-end QoS, there are two potential problems. First, the number of measurements taken is equal to the number of flows with SLA and may not be scalable for a large network. In addition, when problems are detected, locating the problematic links is not straightforward. Additional measurements in the core of the network are still needed. It is precisely these problems that motivated our work.

Finally, the IETF IP performance metrics (IPPM) working group has attempted to develop a set of standard metrics that can be applied to the quality, performance, and reliability of Internet delivery services. For more details, refer to [12].

III. The ARM Framework

This section describes the proposed monitoring mechanism—**ARM**. We first state the assumptions for our study and then outline the QoS measures of interests for SLA flows. After that, we present the framework, discussing how **ARM** incorporates a novel aggregation technique for exchanging measurement data, how the NMS interprets the aggregated data, how the refinement takes place, and when the algorithm terminates. The details of objective-dependent refinement strategies will follow in Section IV.

A. Assumptions

As mentioned in Section I, the **ARM** framework assumes that SLA flows are long lasting, and for QoS reasons the route for each flow is traffic engineered based on technology such as MPLS. The NMS is aware of the route for each individual SLA flow. Changes on such shared information occur at a much slower time scale than that of monitoring sessions. **ARM** also assumes that NMS and each agent maintain the same ordering view of SLA flow identifiers.

We assume that each router can collect packet delay, defined as the time difference between a packet entering and leaving the router. With the current technology, a router can compute this time difference by tagging all incoming packets with a 16-bit timestamp with 1 ms resolution. Such a timestamp allows packet delay for up to 65 s, which should be sufficient for most, if not all, reasonable router performance. Note that this also assumes that the clocks on the interface cards are synchronized to within 1 ms. If a 16-bit timestamp is too expensive, a 8-bit timestamp with 2 ms resolution is another option. For simplicity, we assume zero transmission delay between routers.

We also assume that the per-flow packet arrival count, packet departure count, and packet drop count are all readily available at each router.

Monitoring *session*s are performed periodically (or on demand). During each monitoring session the NMS gathers per-hop QoS data and validates all SLAs. In order to detect and correct SLA violations in time, the interval between periodically performed monitoring sessions should be smaller than the SLA measuring period.

The agent at each router maintains accumulated QoS values over time. However, during each monitoring session, the agent uses the same value recorded at the beginning of the session for reporting throughout the refinement process. In general each session duration is short enough that it is reasonable to assume the accumulated QoS values are relatively stable within each session.

B. QoS Measures for SLA Flows

Typical parameters of an SLA for a flow i include: average throughput (SLA^i_{thr}); end-to-end packet loss ratio ($\mathrm{SLA}^i_{\mathrm{loss}}$); and average end-to-end packet delay ($\mathrm{SLA}^i_{\mathrm{delay}}$). Out of these three parameters, our work has centered around the loss ratio $\mathrm{SLA}^i_{\mathrm{loss}}$ and the delay $\mathrm{SLA}^i_{\mathrm{delay}}$. We assume that the policing at the edge can enforce the average throughput SLA^i_{thr}.

The routers in an ISP network collect measurements of SLA flows passing through them. The router at hop j of an SLA flow i collects its local measurements:

- Loss Ratio ($\mathrm{Local}^{ij}_{\mathrm{loss}}$) = packet drop count of flow i at hop j/packet arrival count of flow i at hop j;
- Average Delay ($\mathrm{Local}^{ij}_{\mathrm{delay}}$) = total packet delay sum of flow i at hop j/packet departure count of flow i at hop j.

Given the local loss ratio and average delay measurements, $\mathrm{Local}^{ij}_{\mathrm{loss}}$ and $\mathrm{Local}^{ij}_{\mathrm{delay}}$, of an SLA flow i at each hop j, the NMS approximates flow i's end-to-end measurements as follows:

$$EtoE^i_{\mathrm{loss}} = \sum_j \mathrm{Local}^{ij}_{\mathrm{loss}} \quad (1)$$

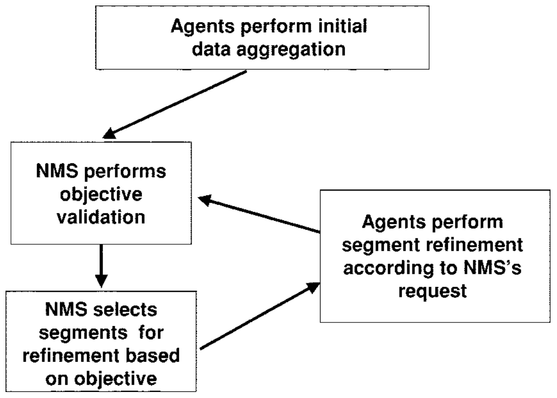

Agents perform initial data aggregation
NMS performs objective validation
Agents perform segment refinement according to NMS's request
NMS selects segments for refinement based on objective

Fig. 1. Execution flow.

$$EtoE^i_{\mathrm{delay}} = \sum_j \mathrm{Local}^{ij}_{\mathrm{delay}}. \quad (2)$$

Note that (1) is a good conservative approximation when loss ratio $\mathrm{Local}^{ij}_{\mathrm{loss}}$ at each router is small. The NMS can calculate the end-to-end loss ratio based on a more precise equation, but that does not affect how the framework operates. The NMS can also add a constant amount to (2) to account for transmission delay between routers.

C. Execution Flow and Functional Components

ARM addresses the scalability and overhead issues in forwarding local measurements to the NMS for end-to-end SLA validation. Fig. 1 shows the execution flow of **ARM**. In each monitoring session:

1) Each agent computes and forwards an aggregation of local measurements to the manager;
2) The manager processes the aggregated data to decide if the measurement objective has been met;
3) **While** the measurement objective is still not met **do**;
4) The manager requests, and the agents respond with refined aggregated data;
5) The manager rechecks the measurement objective based on the refined data.

ARM consists of three major components: dynamic data aggregation, objective validation, and selective refinement. It uses the data aggregation in Steps 1) and 4), the objective validation procedure in Steps 2) and 5), and the selective refinement in Step 4). Every iteration of the agents sending in aggregated data followed by the NMS processing the data is called a *round*. A monitoring session could continue for several refinement rounds until the manager meets its measurement objectives and terminates the session. The following subsections discuss the components in detail. Without loss of generality, the discussion is based on a QoS parameter q (which could be either loss ratio or average delay).

D. Data Aggregation

A naive approach to monitoring the performance of SLA flows is for the NMS to collect performance measurements of each flow from every network device. While this simple polling scheme may be reasonable for a small network, it is inefficient, not scalable, and can cause severe overload as well as congestion at the NMS during each monitoring session.

An enhancement to the simple polling scheme is a simple threshold scheme in which each agent only reports the measurements of flows whose QoS values exceed some thresholds. The problem with such a scheme is the difficulties in selecting the appropriate thresholds dynamically. Setting the threshold too low can easily degenerate it into a simple polling scheme. Setting the threshold too high can cause important data to be overlooked. Worse yet, there is no easy solution to breaking end-to-end QoS requirements into reasonable per-hop threshold at each device. In addition, in times of congestion, information overload can still occur.

The challenge in data aggregation is the tradeoff between data exchange overhead and quality of approximation. We have examined several grouping strategies. One way is to statically assign flows to groups. There is no additional overhead in conveying group membership information at each monitoring session. However, without proper means to predict performance similarity among flows, the static group assignment yields poor approximation. A second approach is to let each agent group flows dynamically based on their QoS values and notify the manger each group's membership along with the aggregated data. Though this approach provides good approximation, the overhead of conveying such membership is now in the same order as that of conveying individual flow data, which defeats the purpose of data aggregation.

ARM uses a data aggregation technique based on *curve approximation* using *segments*. The basic idea is to visualize the set of per-flow q values at each router as a curve, of which the flow identifiers in ascending order is the x axis and the q value is the y axis. The agent at the router then uses a series of bounded segments to approximate a curve. Each segment signifies a group of flows. The upper and lower bound values of each segment represent the maximum and minimum q value of the flows included in the segment. We use three values to encode each segment, the upper bound U, the lower bound L, and the width S (i.e., the number of flows in the segment). As discussed in Section I, since the NMS and each agent share the same ordering view, a sequence of segment width is sufficient to convey the flows contained in each segment.

ARM uses a segment merging algorithm to generate the curve approximation. In the beginning of a data aggregation session, the algorithm makes each individual q value a segment. Next, the algorithm merges selective adjacent segments to form bigger segments. In order to decide which adjacent segments to merge, we compute $\mathrm{difference}(i, i+1)$ between two adjacent segments, which represents the increase in uncertainly if segments i and $i+1$ are merged. Adjacent segments with the smallest difference are merged. The merging process terminates when a desirable number of segments remain.

When two adjacent segments i and $i+1$ are merged, the resulting segment i^{new} carries the encoding $U_{i^{\mathrm{new}}} = \max(U_i, U_{i+1})$, $L_{i^{\mathrm{new}}} = \min(L_i, L_{i+1})$, and $S_{i^{\mathrm{new}}} = (S_i + S_{i+1})$. We define

$$\begin{aligned}\mathrm{difference}(i, i+1) &\\ = [(\max(U_i, U_{i+1}) &- \min(L_i, L_{i+1})) * (S_i + S_{i+1})]\\ &- [(U_i - L_i) * S_i] - [(U_{i+1} - L_{i+1}) * S_{i+1}] \qquad (3)\end{aligned}$$

as the increase in area due to merging, the smaller the better.

Segment Merging Algorithm

```
 **Input Parameters:** a list of $M$ data
 points, an initial aggregation threshold
 $T$, and the maximum number of segments $N$

1  Initialize a series of $M$ segments,
   where each segment corresponds to 1
   data point.
   The upper and lower bound of each seg-
   ment is the data point itself, and the
   segment length is 1.
2  Merge adjacent segments $i$ and $i + 1$ if
   $(U_{i^{\mathrm{new}}} - L_{i^{\mathrm{new}}}) \le T$. Let $C$ be the number of
   segments remaining.
3  while $C > N$ do
4    Select a segment $k$ such that
     $\mathrm{difference}(k, k+1)$ is the smallest;
5    Merge segment $k$ and $k + 1$ and subtract
     $C$ by one.
```

Fig. 2(a) shows a graphical representation of applying our segment merging algorithm to a set of 140 values. The N in this case is 8.

When a network is in normal operating conditions, many flows would have similar loss ratio or average delay. Step 2 merges those data points that are *close enough* in the initial phase to make the algorithm more efficient. The threshold T determines what is considered close enough. For example, flows with difference in packet loss smaller than $1e{-}^{6}$ may be considered close enough.

Note that this algorithm limits the number of output segments to be at most N. In general, the larger the N is, the better the data approximation, though at the cost of additional overhead. A proper choice of N should balance both the data exchange overhead as well as the number of iterations needed to complete a session.

In forwarding the list of segments to the manager, each agent uses a triplet (F, U, L) to represent a segment. The U and L are still upper and lower bounds of QoS values. The F, on the other hand, is the rightmost flow identifier in the segment, which not only defines the segment boundary but also serves as the segment identifier.

E. Objective Validation

Once the manager receives a series of segments for q values from each agent, it must retrieve local QoS values, calculate end-to-end measures, and then validate them against monitoring objectives.

Retrieving local q value of each flow from a series of segments is straightforward. Note that the segment series sent by each agent to the manager approximates a curve, of which the flow identifier is the x axis and the q value is the y axis. The manager knows exactly the set of flows passing through each agent and the flow identifiers. Hence, the manager knows the index of the x axis. With a triplet (F, U, L) for each segment, the manager can compute from left to right along the x axis the upper and lower bound q values of each flow reported in a segment series.

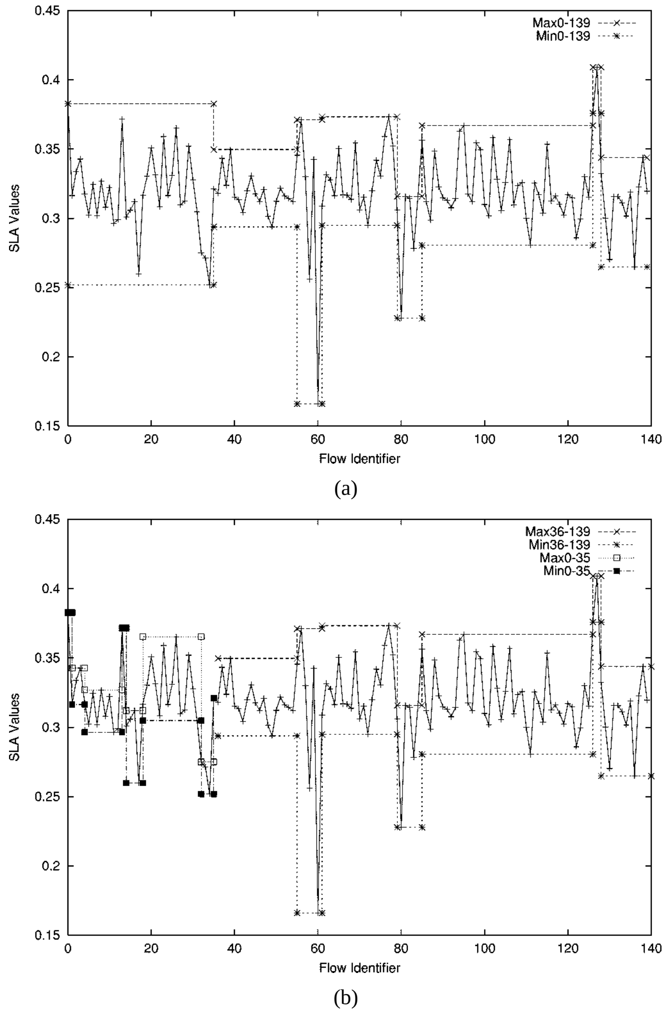


Fig. 2. Example of data aggregation and refinement in ARM.

Calculating end-to-end q measures then follows. By applying (1) or (2), first on the per-hop upper bound and then on the per-hop lower bound, the manager derives the upper and lower bound of the estimated end-to-end q value for each SLA flow.

We proposed three types of objectives, namely: *threshold-based*, *rank-based* and *percentile-based*. With the calculated upper and lower bounds on end-to-end q value for each flow, the NMS then proceeds to validate monitoring objectives. For the threshold-based objectives, it checks the bounded q value against the threshold. For the rank-based and percentile-based objectives, it tries to rank the flows according to their bounded q values.

The bounded q value of some flows may be too loose to help the NMS generate a definite answer. For example, for some flows the threshold may be between the upper and lower bounds. Ranking flows could also be difficult when the bounds of a number of flows overlap. In such cases the NMS must decide which flows need tighter bounds (i.e., refinement) on their q values.

The approaches to selecting flows for refinement are dependent on the types of monitoring objectives. Section IV discusses selection strategies in details. What matters here is that at the end of selection, the NMS identifies a set of flows that require further refinement.

F. Selective Refinement

The purpose of our selective refinement approach is to refine the *coarse* network status pictures that the manager constructed based on reported segment series. As long as there are flows that need refinement, the manager must selectively ask agents to refine reported segments.

Again, to minimize data exchange overhead between the NMS and agents, **ARM** requires that the manager can only ask each agent for refining up to N_{poll} entities, where the entities could be either flows or segments. When the number of flows that need refinement is less than N_{poll}, the NMS can easily just poll the corresponding agents for the exact q value of those flows. Otherwise, the NMS must select not more than N_{poll} segments for refinement. Note that through the refinement rounds in each monitoring session, the NMS and each agent maintain a consistent up-to-date view of segment series.

Manager Selective Request Algorithm

Input Parameters: a list FS of flows in the current round that need refinement, the parameter N_{poll}, and the current segment series H

```
1 if $|FS| \le N_{poll}$ then
2   Poll the agent for the exact $q$ value
    of flows in $FS$;
3 else
4   Let $CS = \{s_i | (FS \cap s_i) \ne \emptyset, s_i \in H\}$ be the
    set of segments in $H$ each of which
    contains at least one flow in $FS$,
    $|CS| = m$;
5   Ask the agent to refine the segments
    in some set $RS \subseteq CS$, where $RS = CS$ if
    $m \le N_{poll}$.
```

The strategies for choosing RS from CS in Step 5 vary. We pick the first N_{poll} segments of CS in our experiments when $m > N_{poll}$. Other strategies could be, for instance, the segments in CS that contain the most flows in need of refinement, or the segments that have the widest range of q values. Note that if $CS = \emptyset$, then the manager does not poll the corresponding agent.

When the manager does polling in Step 2, the agent simply replies with the exact values for the flows listed in FS. Otherwise, the agent performs the following algorithm to refine the requested segments.

Agent Selective Refinement Algorithm

Input Parameters: a list of segments RS in the current segment series to be refined, and the maximum number N_k of new segments to be reported at round k

```
1 Use heuristics to select a number $b_l$
  for refining segment $s_l \in RS$ into $b_l$
  new segments, where $1 \le l \le |RS|$, and
  $\sum_{l=1}^{|RS|} b_l \le N_k$;
2 Apply Segment Merging Algorithm to ob-
  tain a set of new segments $\{s_{li} | 1 \le i \le b_l\}$;
```

```
3  Add a triplet $(f_{li}, U_{li}, L_{li})$ to a reply
   message for each newly created segment
   $s_{li}$;
4  Send the reply message to the manager.
```

The heuristics in choosing b_l in Step 1 tries to strike a balance between maximum increases in total number of segments (so that some flows can get a best approximation quickly) and fair distribution of refinement to all segments (so that more flows can get some value refinement).

For example, assume that a series has N segments, all need refinement. Furthermore, assume that an agent is due to send back $2N$ triplets in reply. Should the agent choose to evenly allocate two to each existing segment, then the manager gets a new series of $2N$ segments, with a moderate refinement on each flow value. However, should the agent choose to refine only one existing segment, assume it is possible, then after receiving $2N$ triplets from the agent the manager now has an updated series of $2N$ new segments plus $N-1$ existing segments (with no refinement) for a total of $3N-1$ segments.

The allocation of b_l is performed as follow. Initially, $b_l = |s_l|$. If $\sum_{l=1}^{|RS|} b_l > N_k$, we randomly decrease some b_l so as to comply with the restriction that $\sum_{l=1}^{|RS|} b_l \leq N_k$. In this process, b_l is set to zero (i.e., no refinement on segment s_l), if it is reduced to less than two. On the other hand, if $\sum_{l=1}^{|RS|} b_l < N_k$, we randomly increase some b_l till the sum equals N_k.

Fig. 2(b) shows the result (of 15 segments total) after refining the first segment in Fig. 2(a) into eight additional segments. Observe that in order to reconstruct the *refined* segment series, the manager only needs the eight triplets $(f_{1i}, U_{1i}, L_{1i}), 1 \leq i \leq 8$ from an agent.

Since we use a triplet to encode each segment, there is intrinsic 50% overhead if we have to report each data point as a segment versus a (flow_id, value) pair.

Corollary 1: Let the number of flows to be reported by an agent be M and the maximum number of segments reported at each round be N. In the worst case, **ARM** needs $(2M/N)-1$ rounds to complete a session, and the total overhead is $3(2M-N)$.

The worst case occurs when we evenly divide N to all segments that need refinement. Thus, in each refinement round an agent can only refine $N/2$ existing segments (breaking them each into two segments) for a total increase of $N/2$ segments in the updated series each round. The agent can report N segments in the initial round. Hence, at round k, the updated segment series would have $X(k) = N + (k-1) \times N/2$ segments. For a complete accurate data on each flow, **ARM** needs $k = (2M/N)-1$ rounds for $X(k)$ be at least M. Since the overhead each agent encounters in a round is $3N$, the total overhead is therefore $3(2M-N)$.

When $M = N$, **ARM** finishes in one round, and the overhead is $3M$ compared with $2M$ of a naive method that reports (flow_id, value) pairs directly. As N becomes smaller, not only **ARM** needs more rounds to complete a session, the worst case overhead approaches $6M$. However, as we will illustrate in Section V, our experimental results show that **ARM** performs much better on average.

IV. Objective-Dependent Selection Strategies for Refinement

This section describes how the NMS determines, based on monitoring objectives, which flows it needs refined QoS values. In the discussions, $EtoE_q^f(\text{upper})_k$ and $EtoE_q^f(\text{lower})_k$ denote, respectively, the upper and lower bound values of a QoS parameter q (loss or delay) calculated at round k for SLA flow f. A key observation through the refinement rounds in a monitoring session is that the upper bound is always monotonic nonincreasing and the lower bound is always monotonic nondecreasing for each flow. That is, $\forall k \geq l \geq 1, EtoE_q^f(\text{upper})_k \leq EtoE_q^f(\text{upper})_l$ and $EtoE_q^f(\text{lower})_k \geq EtoE_q^f(\text{lower})_l$.

A. Threshold-Based Objective

A threshold-based monitoring objective often aims at detecting flows that have violated their end-to-end QoS agreements.

At each round k, $EtoE_q^f(\text{exact})_k$ exists if $EtoE_q^f(\text{upper})_k = EtoE_q^f(\text{lower})_k$. The manager acts in one of the following five cases listed under two broad categories:

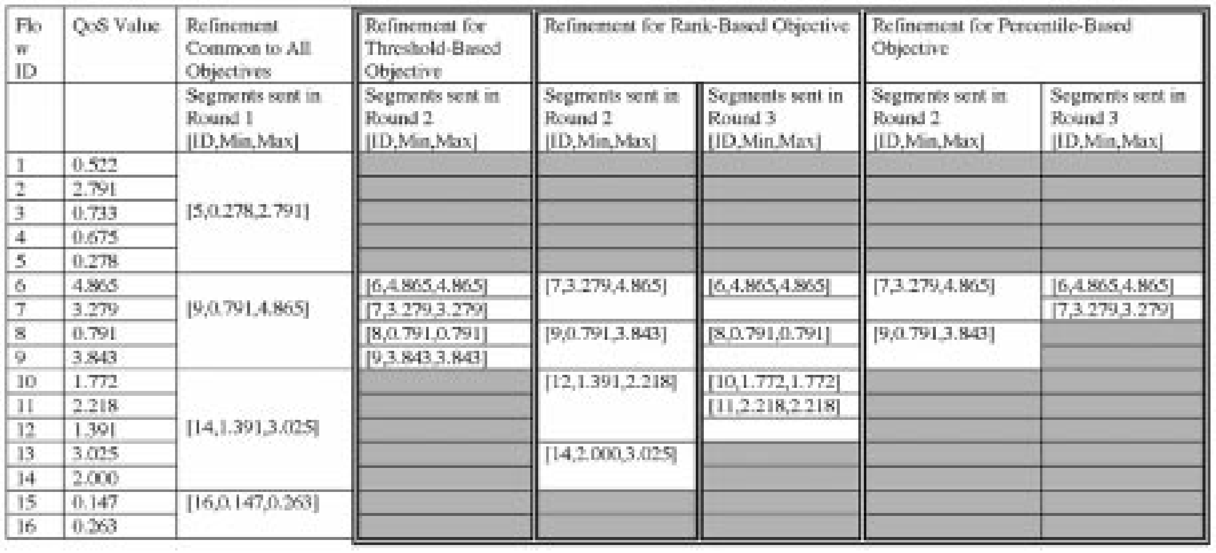
- $EtoE_q^f(\text{exact})_k$ exists, then
  - **Case I:** $EtoE_q^f(\text{exact})_k > \text{SLA}_q^f$. Flow f has violated its q QoS. The manager must take immediate actions to ease the problem.
  - **Case II:** $EtoE_q^f(\text{exact})_k \leq \text{SLA}_q^f$. Flow f is fine for now.
- $EtoE_q^f(\text{exact})_k$ does not exist, that is, some of the reported values for flow f are SLA bounds, then
  - **Case III:** $EtoE_q^f(\text{upper})_k > \text{SLA}_q^f \geq EtoE_q^f(\text{lower})_k$. The manager cannot infer anything definitely in this case.
  - **Case VI:** $EtoE_q^f(\text{lower})_k > \text{SLA}_q^f$. Flow f is definitely in violation of its SLA.
  - **Case V:** $EtoE_q^f(\text{upper})_k \leq \text{SLA}_q^f$. Flow f is fine for now.

In cases II and V, depending on how close it is to a violation, the manager may choose to take some actions such as rerouting the flow. In general, since the manager has per-hop information, it can spot problems at some hops even when the end-to-end measure is fine.

As the objective is to detect possible SLA violations, the only set of flows that require further investigation are those in Case III.

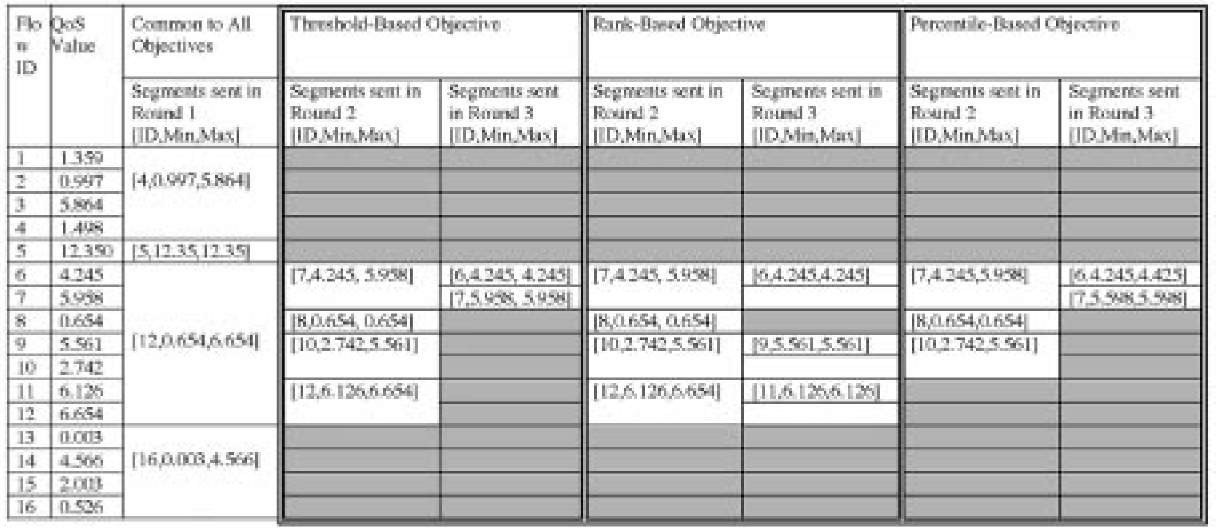
A simple example with 16 SLA flows over two tandem links will be used to illustrate how threshold-based objective works. Figs. 3 and 4 depict the segment data that agents at the first and second hop sent to the manager. Fig. 5 reflects the calculated end-to-end QoS values that the manager maintains. In the example, the number of segments reported by each agent at round k, N_k, is set to a fixed value of four for all k. The QoS values are generated randomly using an exponential distribution with mean 3.

The first three columns of Figs. 3 and 4 show the flow identifiers, the QoS values measured, and the segments sent to the manager by the agents on the first hop and second hop,



Flo w ID	QoS Value	Refinement Common to All Objectives	Refinement for Threshold-Based Objective	Refinement for Rank-Based Objective		Refinement for Percentile-Based Objective	
		Segments sent in Round 1 [ID,Min,Max]	Segments sent in Round 2 [ID,Min,Max]	Segments sent in Round 2 [ID,Min,Max]	Segments sent in Round 3 [ID,Min,Max]	Segments sent in Round 2 [ID,Min,Max]	Segments sent in Round 3 [ID,Min,Max]
1	0.522		▨	▨	▨	▨	▨
2	2.791		▨	▨	▨	▨	▨
3	0.733	[5,0.278,2.791]	▨	▨	▨	▨	▨
4	0.675		▨	▨	▨	▨	▨
5	0.278		▨	▨	▨	▨	▨
6	4.865		[6,4.865,4.865]	[7,3.279,4.865]	[6,4.865,4.865]	[7,3.279,4.865]	[6,4.865,4.865]
7	3.279	[9,0.791,4.865]	[7,3.279,3.279]				[7,3.279,3.279]
8	0.791		[8,0.791,0.791]	[9,0.791,3.843]	[8,0.791,0.791]	[9,0.791,3.843]	▨
9	3.843		[9,3.843,3.843]				▨
10	1.772		▨	[12,1.391,2.218]	[10,1.772,1.772]	▨	▨
11	2.218		▨		[11,2.218,2.218]	▨	▨
12	1.391	[14,1.391,3.025]	▨			▨	▨
13	3.025		▨	[14,2.000,3.025]	▨	▨	▨
14	2.000		▨		▨	▨	▨
15	0.147	[16,0.147,0.263]	▨	▨	▨	▨	▨
16	0.263		▨	▨	▨	▨	▨

Fig. 3. Aggregation and selective refinement hop 1.



Flo w ID	QoS Value	Common to All Objectives	Threshold-Based Objective		Rank-Based Objective		Percentile-Based Objective	
		Segments sent in Round 1 [ID,Min,Max]	Segments sent in Round 2 [ID,Min,Max]	Segments sent in Round 3 [ID,Min,Max]	Segments sent in Round 2 [ID,Min,Max]	Segments sent in Round 3 [ID,Min,Max]	Segments sent in Round 2 [ID,Min,Max]	Segments sent in Round 3 [ID,Min,Max]
1	1.359		▨	▨	▨	▨	▨	▨
2	0.997	[4,0.997,5.864]	▨	▨	▨	▨	▨	▨
3	5.864		▨	▨	▨	▨	▨	▨
4	1.498		▨	▨	▨	▨	▨	▨
5	12.350	[5,12.35,12.35]	▨	▨	▨	▨	▨	▨
6	4.245		[7,4.245, 5.958]	[6,4.245, 4.245]	[7,4.245, 5.958]	[6,4.245,4.245]	[7,4.245,5.958]	[6,4.245,4.425]
7	5.958			[7,5.958, 5.958]				[7,5.598,5.598]
8	0.654		[8,0.654, 0.654]	▨	[8,0.654, 0.654]	▨	[8,0.654,0.654]	▨
9	5.561	[12,0.654,6.654]	[10,2.742,5.561]	▨	[10,2.742,5.561]	[9,5.561,5.561]	[10,2.742,5.561]	▨
10	2.742			▨				▨
11	6.126		[12,6.126,6.654]	▨	[12,6.126,6.654]	[11,6.126,6.126]	▨	▨
12	6.654			▨			▨	▨
13	0.003		▨	▨	▨	▨	▨	▨
14	4.566	[16,0.003,4.566]	▨	▨	▨	▨	▨	▨
15	2.003		▨	▨	▨	▨	▨	▨
16	0.526		▨	▨	▨	▨	▨	▨

Fig. 4. Aggregation and selective refinement hop 2.

respectively, in round 1. Similarly, the first three columns of Fig. 5 show the flow identifiers, the actual end-to-end QoS (computed as the sum of the values on hop 1 and hop 2 using the values along column two), and the QoS approximation after round 1.

In this example, the objective is to find all flows with end-to-end QoS greater than 10. According to the data in column three of Fig. 5, the QoS value of Flow 5 is obviously above 10 since the minimum value is 12.628. Similarly, flows with upper bound below 10 do not need further refinement. The Case III flows are flow 6, 7, 8, and 9. As a result, the manager requested for the agent on hop 1 to refine the segment [9, 0.791, 4.865] and the agent on hop 2 to refine the segment [12, 0.654, 6.654]. After receiving the new (finer) segments in round 2, the only Case III flow is flow 6 (cf. Fig. 5, col. 4). A third round to refine the segment $[7, 4.245, 5.958]$ on link 2 is required to determine that flow 6 has a QoS value below 10.

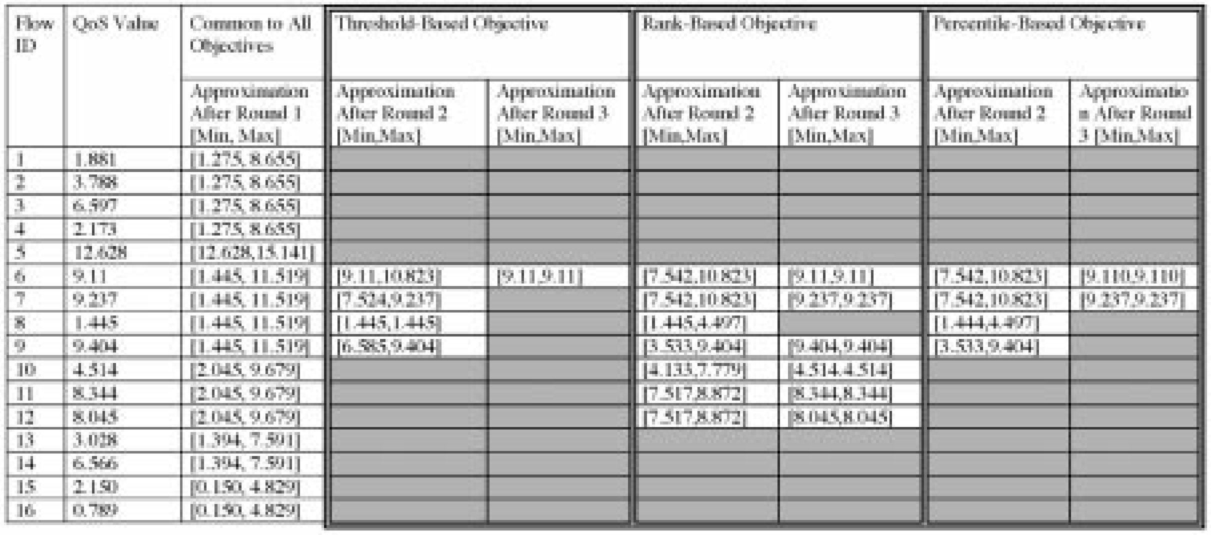
B. Rank-Based Objective

Rank-based objectives identify the N top-ranked flows based on some QoS values, for example, the N flows with the longest average delay. In stating the objective, instead of defining N as a single value, a range will be used. Let (N_L, N_U) represents the range. The NMS considers the objective met when it can identify the top n flows, where $N_L \leq n \leq N_U$.

At round k, let the NMS sorts the entire set of flows it is monitoring based on their end-to-end upper bound of q values to generate an ordered list of flows.

Definition 1: Let F_q^{ik} be the flow that exhibits the ith largest upper bound of QoS parameter q in round k in the ordered list. Let U_q^{ik} and L_q^{ik} be the upper and lower bound end-to-end q values of the flow in position i at round k.

Since $\forall k \geq l \geq 1$, $EtoE_q^f(\text{upper})_k \leq EtoE_q^f(\text{upper})_l$ and $EtoE_q^f(\text{lower})_k \geq EtoE_q^f(\text{lower})_l$, a flow f ranks higher than a flow f' if for any $k > 0$, $EtoE_q^f(\text{lower})_k > EtoE_q^{f'}(\text{upper})_k$.



Flow ID	QoS Value	Common to All Objectives	Threshold-Based Objective		Rank-Based Objective		Percentile-Based Objective	
		Approximation After Round 1 [Min, Max]	Approximation After Round 2 [Min,Max]	Approximation After Round 3 [Min,Max]	Approximation After Round 2 [Min,Max]	Approximation After Round 3 [Min,Max]	Approximation After Round 2 [Min,Max]	Approximatio n After Round 3 [Min,Max]
1	1.881	[1.275, 8.655]	▨	▨	▨	▨	▨	▨
2	3.788	[1.275, 8.655]	▨	▨	▨	▨	▨	▨
3	6.597	[1.275, 8.655]	▨	▨	▨	▨	▨	▨
4	2.173	[1.275, 8.655]	▨	▨	▨	▨	▨	▨
5	12.628	[12.628,15.141]	▨	▨	▨	▨	▨	▨
6	9.11	[1.445, 11.519]	[9.11,10.823]	[9.11,9.11]	[7.542,10.823]	[9.11,9.11]	[7.542,10.823]	[9.110,9.110]
7	9.237	[1.445, 11.519]	[7.524,9.237]	▨	[7.542,10.823]	[9.237,9.237]	[7.542,10.823]	[9.237,9.237]
8	1.445	[1.445, 11.519]	[1.445,1.445]	▨	[1.445,4.497]	▨	[1.444,4.497]	▨
9	9.404	[1.445, 11.519]	[6.585,9.404]	▨	[3.533,9.404]	[9.404,9.404]	[3.533,9.404]	▨
10	4.514	[2.045, 9.679]	▨	▨	[4.133,7.779]	[4.514,4.514]	▨	▨
11	8.344	[2.045, 9.679]	▨	▨	[7.517,8.872]	[8.344,8.344]	▨	▨
12	8.045	[2.045, 9.679]	▨	▨	[7.517,8.872]	[8.045,8.045]	▨	▨
13	3.028	[1.394, 7.591]	▨	▨	▨	▨	▨	▨
14	6.566	[1.394, 7.591]	▨	▨	▨	▨	▨	▨
15	2.150	[0.150, 4.829]	▨	▨	▨	▨	▨	▨
16	0.789	[0.150, 4.829]	▨	▨	▨	▨	▨	▨

Fig. 5. Manager's view of end-to-end QoS.

The approach to validating rank-based objective is to break the ordered set of flows into several distinct subset of flows such that flows in one subset is always ranked lower or higher than flows in a different subset. More precisely, if a set of flows $\mathcal{F}$ is divided into m ordered subsets $\{\mathcal{F}_j\}$, $1 \leq j \leq m$, and $n_j = |\mathcal{F}_j|$ be the number of flows in $\mathcal{F}_j$, then the first n_1 flows belong to $\mathcal{F}_1$, the next n_2 flows belong to $\mathcal{F}_2$, etc. We name each boundary position in the list a *delimiter index*. For example, n_1 is a delimiter index which ends the block of $\mathcal{F}_1$, so is $(n_1 + n_2)$ which ends the block of $\mathcal{F}_2$. The task of validating a rank-based objective for QoS q now becomes searching for a delimiter index at position i such that $N_L \leq i \leq N_U$.

Note that through the refinement rounds in each monitoring session, a flow f may not occupy the same position in the ordered list due to changes in $EtoE_q^f(\text{upper})_k$.

Definition 2: The *delimiter index* D_q^{ik} exists if in round k $\forall j$, $1 \leq j \leq i < l$, $L_q^{jk} > U_q^{lk}$. The set of flows delimited by D_q^{ik}, denoted S_q^{ik}, is the set $\{F_q^{jk} | 1 \leq j \leq i\}$.

Theorem 1: If D_q^{ik} exists, than $\forall k' > k$, $D_q^{ik'}$ exists, and $S_q^{ik'} = S_q^{ik}$.

These delimiter indexes exhibit a nice property stated in Theorem 1, which greatly helps the refinement steps. Simply, once the manager identifies a delimiter index D_q^{ik} in round k, then the flows in S_q^{ik} will always be ranked higher than any other flow $f \notin S_q^{ik}$ in all subsequent refinement rounds. That is, when $i < N$, the top N flows must at least include all the flows in S_q^{ik}. Hence, the manager can concentrate on requesting refinements on flows not in S_q^{ik}. See Appendix for the proof of Theorem 1.

Each monitoring session of a rank-based objective works in the following way. In each round, the NMS calculates upper and lower bound end-to-end QoS values and sorts the flows according to their upper bound QoS values. It then searches delimiter indexes based on the ordered list. At each round k, if $\exists D_q^{ik}$, such that $N_L \leq i \leq N_U$, then S_q^{ik} is the set of flows that meet the objective, and the session terminates. Otherwise, there is a pair of consecutive delimiter indexes D_q^{ik} and D_q^{jk} such that $i < N_L \leq N_U < j$. Clearly, the NMS needs refined data on the set of flows $(S_q^{jk} - S_q^{ik})$ in order to identify more delimiter indexes between positions i and j in the subsequent rounds and to meet the termination condition.

A further optimization is performed since j can be much larger than i. Due to the limited number of segments per refinement, it is more efficient to focus on flows closer to the target rank. Define a slack factor P_{slack}, $0 < P_{\text{slack}} \leq 1$. For each round, let the number of flows to be refined equal to $\min\{1, P_{\text{slack}} * (j - i)\}$ flows. These flows are selected starting from the flow right after F_q^{ik}. At least one flow must be refined to ensure progress.

The same setup used to illustrate threshold-based objective in Section IV-A is used to illustrate rank-based objective. The objective is to find flows with the three largest QoS values. P_{slack} is set to 0.3. From column three of Fig. 5, there are only two delimiter indexes after round 1, $D_q^{(1)(1)}$ and $D_q^{(16)(1)}$. Flow 5 is rank one, the flow with the largest QoS value. In round 2, the manager requested refinement for flows 6 to 10, the flows with the next five largest U_q^{i1}, $1 < i \leq 16$. Similarly, in round 3, flows 6, 7, 9, 11,m and 12 are refined. Finally, at the end of round 3, the ordered list based on upper bound values is [5, 9, 7, 6, 1, 2, 3, 4, 11, 12, 13, 14, 15, 16, 10, 8]. Three more flow delimiter indexes are identified, $D_q^{(2)(3)}$, $D_q^{(3)(3)}$, and $D_q^{(4)(3)}$. Flows 5, 9, and 7 can now be easily identified as the 3 flows with the largest value. The NMS needs not know the exact ordering of the rest of flows.

C. Percentile-Based Objective

We consider two kinds of percentile-based monitoring objectives in our study. One is to validate the QoS value of some percentile of flows, such as "Do $X\%$ of flows in the network have loss ratio less then a value Y?" The monitoring terminates when the NMS can decide either $X\%$ of loss ratio upper bounds are less than Y (the statement is true) or $(1 - X)\%$ of loss ratio lower bounds are greater than Y (the statement is false).

The second kind is to find the QoS value of the Xth percentile flow in the network accurate to within a specific range. The NMS should report that the Xth percentile flow exhibits a QoS value Y, where the gap between the upper and lower bounds of Y is sufficiently small.

Note that rank-based objective is different from percentile-based objective in that the former looks for the list of top Nth flows while the later looks for a bounded QoS value.

Let $T(X)$ be the flow position corresponding to the target percentile X. For example, if there are 1000 flows, and the target is the 98th percentile, then $T(98\mathrm{th}) = 20$.

To address percentile-based objectives, the NMS sorts the entire set of flows twice to generate two separate lists: one is based on their upper bound values; the other, their lower bound values. In Section IV-B, we refer to the ith flow and its upper bound q value in the upper-bound ordered list at the round k as F_q^{ik} and U_q^{ik}. Here we add the notations that for the lower-bound ordered list the ith flow and its lower bound q value at round k are $F_{L_q}^{ik}$ and $L_{L_q}^{ik}$, respectively.

Let E_q^i be the ith largest exact end-to-end q value. Theorem 2 provides a nice property to relate the two ordered lists of QoS values. In successive refinement rounds, Theorem 2 states that the gap between U_q^{ik} and $L_{L_q}^{ik}$ shrinks and eventually converges toward the exact QoS value of the ith flow, E_q^i. Note that F_q^{ik} and $F_{L_q}^{ik}$ need not be the same flow at round k. Consequently, for all i and k, $L_{L_q}^{ik} \leq E_q^i \leq U_q^{ik}$.

Theorem 2: For any $i, \forall k, l, l > k > 0$, $L_{L_q}^{ik} \leq L_{L_q}^{il} \leq E_q^i \leq U_q^{il} \leq U_q^{ik}$.

With Theorem 2, the task of dealing with percentile-based objectives is straightforward. For the first kind of percentile-based objectives, the NMS checks at round k the QoS values at the position $p = T(X)$.

- **Case I:** $E_q^i \leq U_q^{pk} \leq Y$. The objective is validated.
- **Case II:** $E_q^i \geq L_{L_q}^{pk} > Y$. The objective has failed.
- **Case III:** $U_q^{pk} > Y \geq L_{L_q}^{pk}$. The verdict is still out. A flow i should be refined in the next round if $U_q^{ik} > Y \geq L_{L_q}^{ik}$.

For the second percentile-based objective, the NMS first determines if the gap between U_q^{pk} and $L_{L_q}^{pk}$ is sufficiently small. If an objective requires a tighter bound, the NMS searches a pair of consecutive delimiter indexes, D_q^{ik} and D_q^{jk}, as described in Section IV-B for monitoring rank based objectives, such that $i < p < j$. The set of flows $NS = (S_q^{jk} - S_q^{ik})$ include the flow f that should eventually end up at position p through refinement rounds. The NMS also knows that whichever flow f is, its q value is bounded by U_q^{pk} and $L_{L_q}^{pk}$. Hence, it can request refinement on all flows in NS except those flows $f' \in NS$ such that either $EtoE_q^{f'}(\mathrm{upper})_k < L_{L_q}^{pk}$ or $EtoE_q^{f'}(\mathrm{lower})_k > U_q^{pk}$.

Again, the same setup is used to illustrate percentile-based objective. The objective here is to find out if the QoS value of the third-highest flow is above or below ten. At the end of round 1, from column three of Fig. 5, all flows i such that ranges do not include ten are eliminated and the third-highest flow has QoS value between 2.045 and 11.519. Flows that required refinement are flow 6 to 9. Data obtained in round 2 (column eight of Fig. 5) narrows the range to between 7.542 and 10.823. Data from round 3 (column nine of Fig. 5) determines that the third-highest flow has the QoS value 9.237. Notice that using Theorem 2, this conclusion can be drawn without getting a smaller range for flow 9, which is between 3.533 and 9.404.

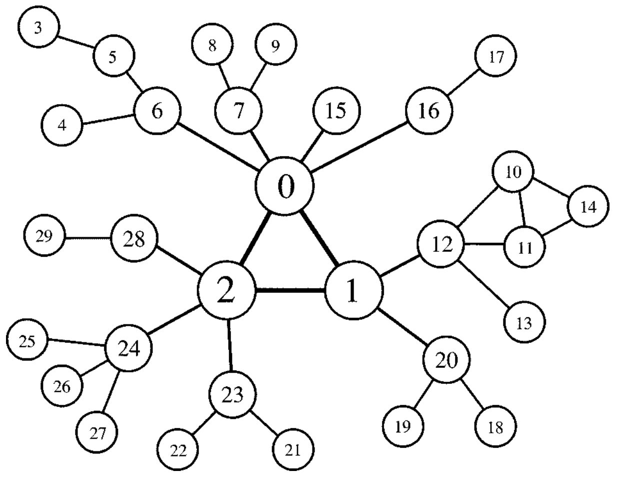


Fig. 6. Simulation network topology.

V. Experimental Results

In order to evaluate the effectiveness of the proposed algorithm in monitoring the service performance of a network with QoS guarantees, we conducted extensive experiments in a simulated network domain. The result reported in this section addresses the following issues:

- the advantage of using the proposed monitoring algorithm in terms of overhead reduction;
- the tradeoff between overload and iteration time;
- the effect of changing N_k, where N_k is the maximum number of new segments each agent reports at round k (cf. the Agent Selective Refinement Algorithm).

Only results for threshold-based and rank-based measurements are presented.

A. Testbed Setup

The experiments were carried out over a randomly generated 30 nodes topology shown in Fig. 6.

The topology is organized as a single three level hierarchy. The highest level is the core routers consisting of nodes 0, 1, and 2. The next level routers are nodes 6, 7, 12, 15, 16, 20, 23, 24, and 28. The rest are edge routers.

All links are duplex. The one-way bandwidth of each link depends on the type of routers it connects at both ends. It is 20 Mb/s for links connecting two core routers, 15 Mb/s for ones between a core and a next level router, and 10 Mb/s for the rest.

All experiments are performed using ns-2. An on–off model is used to generate traffic with different average rate and burst size. Leaky bucket is used for policing at the edge routers. Input traffic is selected from the four classes listed in Table I, which shows the leaky bucket parameters associated with each traffic class used in the simulations. All flows in the simulation have the same SLA, which allows average end-to-end delay of 150 ms and loss ratio of 0.02. Within the network, packets are scheduled using the first-in-first-out (FIFO) discipline.

There is a local network management agent on each router. A centralized network manager collects aggregated data from these agents. For simplicity, in our simulation we placed a

TABLE I
TRAFFIC PARAMETERS FOR THE FOUR CLASSES USED IN THE SIMULATIONS

	Class 1	Class 2	Class 3	Class 4
r (kb/s)	128	128	64	64
busy time (s)	0.3	0.5	0.3	0.5
idle time (s)	0.7	0.5	0.7	0.5

management link between each node and the central network manager. Loss ratio and average delay were collected at each network node and samples of the statistics were reported to the manager periodically based on the algorithm presented in Sections III and IV.

B. Random Load Generator

We developed a random load generator to generate network traffic with various loading conditions. The load generator mimics admission control procedures in practice. It runs a flow generation loop. For each iteration in the loop, it randomly selects two edge routers as the source-destination pair for a flow, and selects traffic class for the flow. The generator then attempts to "admit" the flow by securing its resources (in this case, *average bandwidth*) along its route.

To create overload situations on some number of links, flows are admitted even when there is insufficient link bandwidth along portion of the path. Nevertheless, a list of links that have been "over-subscribed" is maintained. The flow generation loop terminates when the number of admitted flows is at least X and the number of over-subscribed links is more than Y.

The above steps result in reasonable traffic pattern variations, but only within a range of overload conditions. In order to generate a wide range of network load, where the number of flows violating their SLAs varies from none to almost the entire set of flows, admission control is performed changing the *overload factor* of each link. A link is now "over-subscribed" if the total average throughput of flows admitted is less than or equal to the product of overload factor and link bandwidth.

The overload factor reflects how willingly an ISP wants to risk SLA violations. The smaller the factor, the more conservative the admission control is, and the lesser SLA violations the network may observe.

We set $X = 1000$ and $Y = 8$ for all traffic loads generated in our experiments. The overload factor is varied from 0.5 to 1.5.

C. Comparison of Monitoring Performance Using Threshold-Based Objective

The performance of our threshold-based monitoring scheme is compared with two centralized off-line schemes which are expected to perform well. In both schemes, it is assumed that all flow status are known by a single *virtual* management agent and this virtual agent only sends to the network manager data relating to flows with SLA violations. In *scheme-1*, for each flow with SLA violation the virtual agent sends to the manager QoS data of the flow collected at all hops. In *scheme-2*, instead of sending data collected at all hops for those flows with SLA violations, only sufficient information is sent such that the manager can confirm their violation status. That is, if there are 3 hops and a significant loss is occurring only on a single congested link, then only the loss ratio on that link is sent. This is the minimum information required to identify a SLA violation without resorting to some form of aggregation.

Comparison is based on the total count of all data items sent from the agents to the manager. Each data item, regardless of its type, has a count of one. For the idealized schemes, each update consists of two data items, one for the flow identifier and the other for the measured value. For **ARM**, the overhead for one update is three (maximum value, minimum value, and flow identifier). In addition, a minimum overhead of two data items is incurred in all **ARM** message exchanges to indicate the number of delay and loss updates.

Let I be the set of flows with loss violation in a session, and J be the set of flows with delay violation in the same session.

- For scheme-1, the count per session is $2 \times (\sum_{i \in I} Hop_i + \sum_{j \in J} Hop_j)$, where Hop_i is the hop count of flow i.
- For scheme-2, the count per session is $2 \times (\sum_{i \in I} Min_i + \sum_{j \in J} Min_j)$ where Min_i is the minimum number of hops to decide if violation occurs in flow i.
- For **ARM**, the count per session is $\sum_{r=1}^{R}(2 + \sum_{k=1}^{K}(3 \times (H_{rk}^{\text{loss}} + H_{rk}^{\text{delay}}) + 2 \times (P_{rk}^{\text{loss}} + P_{rk}^{\text{delay}})))$ where $R =$ number of rounds, $K =$ number of links, and, for each link k in round r, $H_{rk}^{\text{loss}} =$ number of new loss segments, $H_{rk}^{\text{delay}} =$ number of new delay segments, $P_{rk}^{\text{loss}} =$ number of loss polling updates, and $P_{rk}^{\text{delay}} =$ number of delay polling updates.

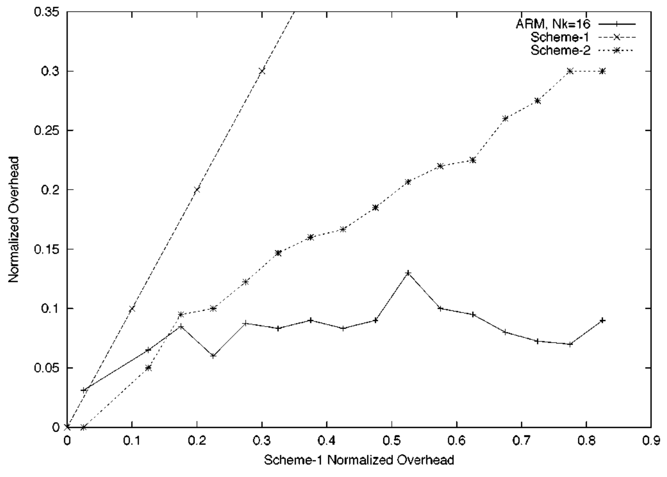
For comparison, a simple polling approach requires a total count of $4 \times (\sum_{i \in F} Hop_i)$, where F is the entire set of flows, and Hop_i is the hop count of flow i. Each entry consists of the two pairs (flow_id, loss value) and (flow_id, delay value). Note that if the manager and agents share the sorted list of flows, as we have assumed for **ARM**, a better polling approach will be to send only the sorted QoS data without flow identifiers. Nevertheless, it will only change the normalized values reported in our experimental results, but not the relative measures between **ARM** and the other two idealized schemes.

All experiments ran for 100-s simulation time excluding a five seconds warmup time. The performance of various algorithms are evaluated by running each algorithm 50 times using different traffic loads generated by the random load generator. The minimum number of flows in an experiment is 1000, the maximum is 1864 and the average is 1306. The minimum total data item count using simple polling in an experiment is 12 944, the maximum is 25 868 and the average is 18 344.

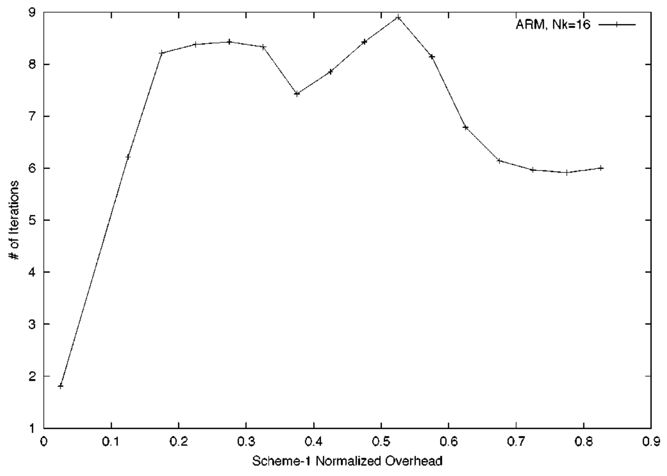
In the first experiment, the parameter N_k is fixed at 16 for the entire simulation run. N_{poll} is fixed at 32 for all experiments.

The measurement overhead of all three schemes are normalized by dividing the monitoring data item count by the total count required in a simple polling approach. That is, if the count for simple polling is 1000 and the count for **ARM** is 100, the normalized overhead for **ARM** is $100/1000 = 0.1$.

Fig. 7 shows the performance of scheme-1, scheme-2 and **ARM** relative to that of scheme-1 for the same traffic load. The x axis is the normalized scheme-1 overhead, and the y axis is the respective normalized overhead of scheme-1, scheme-2, and **ARM**. The choice of scheme-1 normalized overhead for the x axis serves as an indication of the number of SLA violations in the network, though the relationship is not exact because the

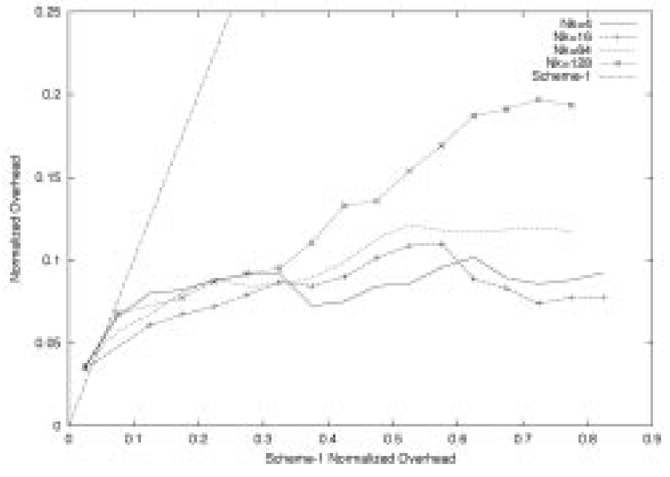


(a)

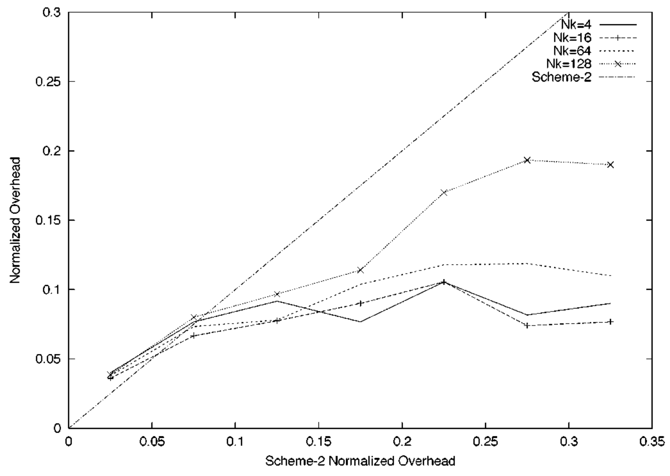


(b)

Fig. 7. Performance comparison between scheme-1, scheme-2, and ARM.



(a)



(b)

Fig. 8. Comparison of scheme-1, scheme-2, and ARM for $N_k = 4,\ 16,\ 64,$ and 128.

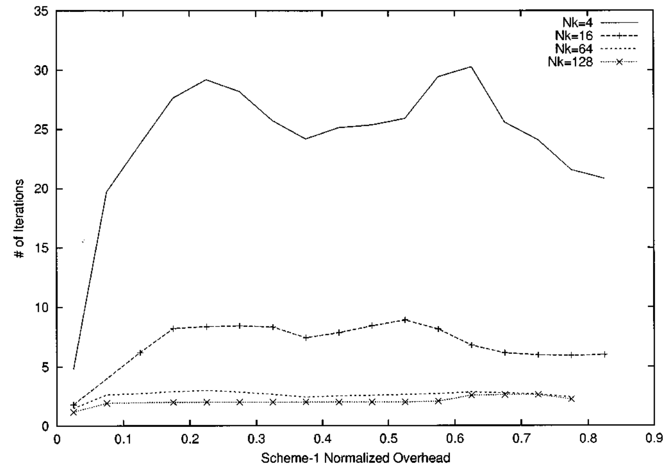
monitoring overhead also depends on the number of hops the flows go through. To make the data easier to read, for each scheme we display only the mean value of the data within each 0.05 segment along the x axis. In other words, the value depicted at $x + 0.025$ corresponds to the mean of all values collected within the segment x to $x+0.05$. As an example, in Fig. 7(a) the set of traffic loads that generates average normalized overhead between 0.45 to 0.5 using scheme-1 generates average normalized overhead of 0.09 using **ARM** with $N_k = 16$.

When there is no SLA violation, **ARM** incurred a minimum normalized overhead of 0.02, whereas, scheme-1 and scheme-2 have no overhead. However, as the number of SLA violations increases, normalized overhead for **ARM** increases slowly and performs better than scheme-1 for normalized overhead larger than 0.06. Beyond normalized overhead of 0.15, **ARM** performs even better than scheme-2. This may come as a surprise since scheme-1 and scheme-2 are highly optimized schemes with very low redundant information exchanged. The difference is that in these two cases, exact values are exchanged. On the other hand, **ARM** provides only bounds on these values and can, thus, aggregate many values into a single segment. Another advantage of **ARM** is that as the number of violations increases, the normalized overhead does not increase linearly with the number of violations. It is due to the fact that once the lower bound of the QoS values violates the SLA, the computation can terminate and there is no need to obtain the actual values.

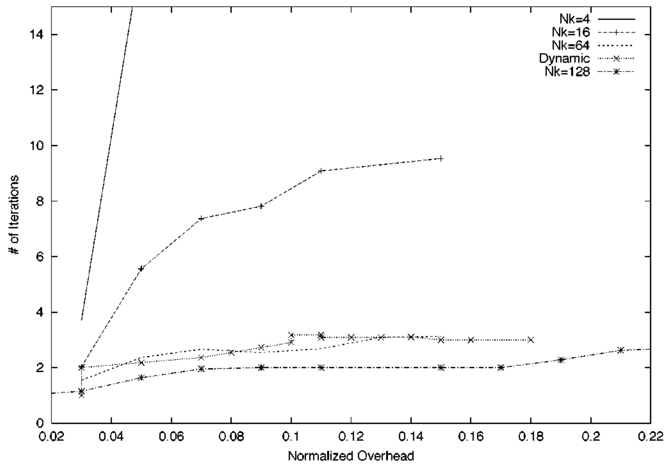
Fig. 7(b) shows the average number of iterations it takes before **ARM** terminates using the same x axis segments and y axis averages. When the number of violations is small, it takes much longer to detect all violations because it is harder to aggregate values and a much finer picture of the network is needed before SLA validation can be completed. However, as the number of violations increases, it becomes easier to detect violations as aggregation of *similar* values becomes more common.

Fig. 8(a) shows the improvement of **ARM** over scheme-1 and Fig. 8(b) shows the improvement of **ARM** over scheme-2 for N_k set to 4, 16, 64, and 128. Fig. 9(a) shows the average number of rounds for N_k set to 4, 16, 64, and 128. In these experiments, N_k is fixed during a single simulation run. The same segment size of 0.05 is used on the x axis, so is the segment mean value on the y axis.

Fig. 8(a) and (b) show that the overhead incurred by **ARM** increases with N_k. This is because when N_k is large, the number of measurements collected in each round may be much more



(a)



(b)

Fig. 9. Performance of ARM and dynamic ARM for $N_k = 4, 16, 64$, and 128.

than what is needed. The extent of such excessive measurement increases as N_k gets larger. The average overhead incurred when $N_k = 128$ is about twice the amount of overhead incurred when $N_k = 16$. A smaller N_k is more efficient. On the other hand, when N_k is too small, the amount of new information collected in each round may be too little and many rounds are needed before the algorithm can terminate. In Fig. 9(a), the average number of rounds required when $N_k = 4$ is about ten times that required when $N_k = 128$. A larger N_k can, thus, lead to much shorter termination time.

The tradeoff in our scheme is between data collection overhead and termination time. Fig. 9(b) shows a plot of the normalized overhead vs. number of round for various values of N_k that clearly illustrates this tradeoff. The figure indicates that decreasing N_k decreases the basic overhead of the algorithm but at the same time increases the number of rounds it takes for the algorithm to converge. On the other hand, increasing N_k to 128 keeps the number of rounds to a very small value but increases the normalized overhead. In addition, since more data are sent in a single cycle, a larger N_k increases the load at the network manager. Thus, N_k should not be set beyond some threshold in order to avoid degenerating **ARM** into a simple polling scheme.

D. Comparison of Monitoring Performance Using Rank-Based Objective

In this section, the experiments focus on two issues:

- How does the performance of rank-based **ARM** scale with load?
- How does the performance of rank-based **ARM** vary with the slack parameter?

In all the results shown in this section, each point corresponds to an experimental run and the dotted line corresponds to the average value for ten runs. The objective in each experiment is to find the top N flows with the largest loss and delay measure, where N is any value between 5 to 20.

Fig. 10(a) shows the normalized overhead of rank-based **ARM** with the overload factor varies from 0.7 to 1.4. The slack factor is set to 0.1. Therefore, in each round, only ten flows closest to the first delimiter is refined. The segment size, N_k, is set to 32.

The result shows that the normalized monitoring overhead increases as the load increases, though the overhead of **ARM** is still significantly lower than simple polling. As the load increases, the number of flows with high loss and delay measure also increases making it harder to differentiate the top N flows. Unlike the threshold-based objective, at very high load, the normalized overhead keeps increasing.

Figs. 10(b) and 11(a) show the impact of varying the slack parameter from 0.01 to 1.0.

With a very small P_{slack} value of 0.01, only a small number of flows are refined per round and the result is lower overhead but may result in longer termination time. When P_{slack} is set to 1.0, the entire set of the eligible flows is passed to the selective refinement process.

The sudden increase in termination time in Fig. 11(a) is due to the number of flows that required refinement per round exceeding the number of the segment N_k added each round. In Fig. 11(b), when a larger N_k of 64 is used, the sudden increase in termination time occurred at a large P_{slack} value.

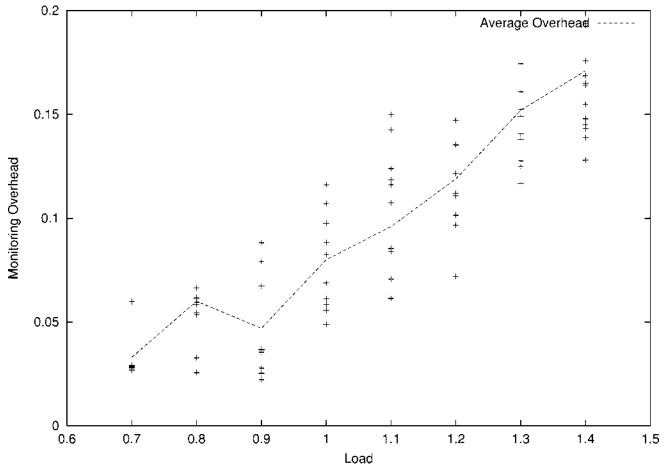
From the result shown, a slack factor of 0.1 to 0.2 seems to have the best tradeoff between overhead and termination speed.

E. Summary of Results

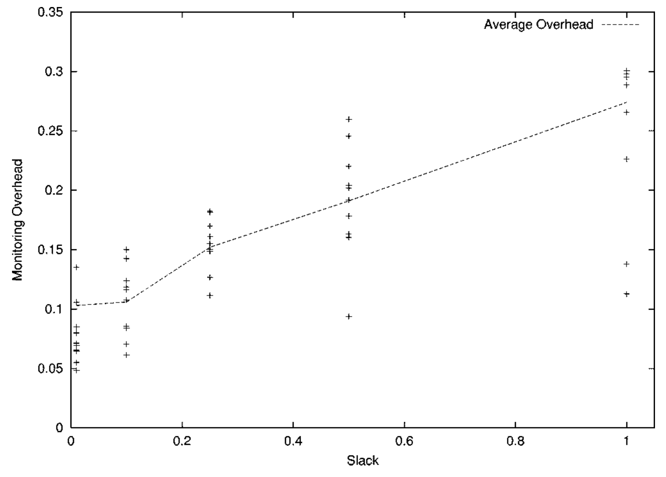
The results in the section can be summarized as follows:

- **ARM** is very efficient and performs well over a wide range of traffic load. In Section V-C, **ARM** is shown to perform better than scheme-1 and scheme-2 in most cases except where the number of violations is very low. This is true for all N_k shown.
- A small N_k reduces the normalized overhead but increases the number of round required. The reverse is also true.
- The effect of the slack factor P_{slack} in Section V-D is similar to that of N_k, the maximum segment size per refinement. The difference is that P_{slack} is specific to the rank-based objective and takes into account the rank of the flows after sorting. On the other hand, N_k is objective independent.

Before concluding this section, it is important to point out that while the performance of **ARM** is fairly robust over a wide range of traffic load, the quantitative result of **ARM** may

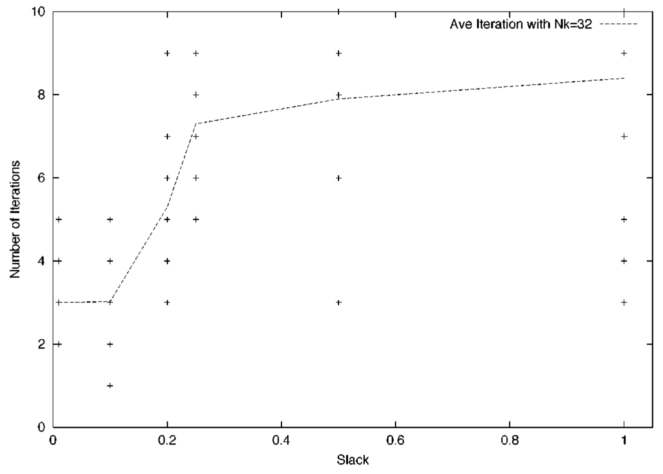


(a)

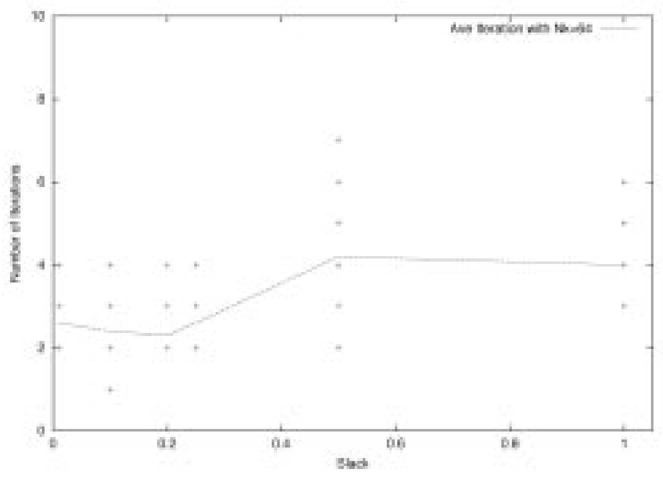


(b)

Fig. 10. Performance of rank-based monitoring.



(a)



(b)

Fig. 11. Termination time of rank-based monitoring with overload factor = 1.1.

change if the total number of flows in the network changes by more than an order of magnitude. Hence, while **ARM** allows significant portion of the monitoring process to be automated, in order to optimize the performance, the value of N_k still requires some tuning depending on the number of flows in the network.

VI. Conclusion

We have presented a generic monitoring framework, **ARM**, to address the scalability in monitoring network QoS that can be configured to run with different objectives. The monitoring is based on hop-by-hop measurement of QoS values, aiming at deriving qualitative status of flows and links that can quickly builds up a coarse picture of the network status that can be refined as required. With a dynamic data aggregation technique and an iterative refinement process, the proposed framework, **ARM**, achieved substantial reduction in overhead and scaled well over a wide range of traffic loads.

Two future directions are of immediate interests to us. We plan to look at the monitoring issues with routers that employ more sophisticated queuing mechanisms such as weighted fair queuing (WFQ) or support differentiated services. We also plan to look into ways of using the monitoring results to trigger management actions. In particular, we can easily extend **ARM** to identify not only flows that violate their SLAs, but also those that receive significantly better services than what their SLA stated. Based on such a monitoring tool we plan to develop an SLA management application to adjust provisioning among these flows. After all, it is a provider's best interest to utilize available resources to satisfy as many SLA flows as possible.

Appendix
Proofs

To prove Theorem 1, we present an interesting property about the curve representing the sorted upper bound QoS values. Lemma 1 says that the curve can only move downwards across the entire domain, regardless the shuffling of flow orders in refinement rounds.

Lemma 1: $\forall i,\, k,\, U_q^{i(k+1)} \leq U_q^{ik}$.

Proof: We prove it by cases.

- **Case I:** F_q^{ik} and $F_q^{i(k+1)}$ are the same flow. Since for every flow f, $EtoE_q^f(\text{upper})_{k+1} \leq EtoE_q^f(\text{upper})_k$, we have $U_q^{i(k+1)} \leq U_q^{ik}$.

- **Case II:** $F_q^{i(k+1)}$ was F_q^{jk}, where $j > i$. That is, $U_q^{jk} \leq U_q^{ik}$. Again, due to the monotonic nonincreasing property of upper bound values, $U_q^{i(k+1)} \leq U_q^{jk}$. Hence, $U_q^{i(k+1)} \leq U_q^{jk} \leq U_q^{ik}$.
- **Case III:** $F_q^{i(k+1)}$ was U_q^{mk}, where $m < i$. That is, $U_q^{ik} \leq U_q^{mk}$. For F_q^{mk} to move down to the ith flow in the sorted list of round $k+1$, some flow $F_q^{m'k}$, $m' \geq i$, must have become $F_q^{m''(k+1)}$, $m'' < i$. Moreover, $U_q^{m''(k+1)} \leq U_q^{m'k}$ for the same monotonic nonincreasing property. Since, $m'' < i$, $U_q^{i(k+1)} \leq U_q^{m''(k+1)}$. Similarly, since $m' \geq i$, $U_q^{m'k} \leq U_q^{ik}$. We now have $U_q^{i(k+1)} \leq U_q^{m''(k+1)} \leq U_q^{m'k} \leq U_q^{ik}$.

□

Theorem 1: If D_q^{ik} exists, than $\forall\, k' > k$, $D_q^{ik'}$ exists, and $S_q^{ik'} = S_q^{ik}$.

Proof: We prove for $k' = (k+1)$. Once the proof is in place, induction takes its own course for any $k' > k$. By Lemma 1, we have $U_q^{i(k+1)} \leq U_q^{ik}$. For every flow $f \in S_q^{ik}$, by Definition 2 $EtoE_q^f(\text{upper})_k \geq EtoE_q^f(\text{lower})_k > U_q^{(i+1)k}$. Hence, $EtoE_q^f(\text{upper})_k \geq EtoE_q^f(\text{upper})_{(k+1)} \geq EtoE_q^f(\text{lower})_{(k+1)} \geq EtoE_q^f(\text{lower})_k > U_q^{(i+1)k} \geq U_q^{(i+1)(k+1)}$. Let DS be the set of flows $\{F_q^{j'(k+1)} | 1 \leq j' \leq i\}$. Since the upper bound of flow f in round $k+1$ is greater than $U_q^{(i+1)(k+1)}$, $f \in DS$. Therefore, $S_q^{ik} \subseteq DS$. However, since the cardinality of both sets are the same (i), we have $S^{ik} = DS$. Because $EtoE_q^f(\text{lower})_{(k+1)} > U_q^{(i+1)(k+1)}$ for every flow $f \in S_q^{ik} = DS$, we conclude that $D_q^{i(k+1)}$ exists, and DS is $S_q^{i(k+1)}$. □

We can also prove a lemma about the curve representing the sorted lower bound QoS values. Lemma 2 says that the curve can only move upwards across the entire domain.

Lemma 2: $L_{L_q}^{i(k+1)} \geq L_{L_q}^{ik}$.

Proof: Prove by cases similar to that for proving Lemma 1. We omit details. □

Theorem 2: For any $i, \forall\, k, l, l > k > 0$, $L_{L_q}^{ik} \leq L_{L_q}^{il} \leq E_q^i \leq U_q^{il} \leq U_q^{ik}$.

Proof: It simply follows Lemma 1 and Lemma 2. □

Acknowledgment

The authors would like to thank the anonymous reviewers for critical comments on previous versions of this paper, which helped them improve the presentation. X. Wang contributed to the initial study of the problem, including a preliminary setup of simulation.

References

[1] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," Internet Engineering Task Force (IETF), RFC 1157, May 1990.

[2] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser, "Protocol operations for version 2 of the simple network management protocol (SNMPV2)," Internet Engineering Task Force (IETF), RFC 1905, Jan. 1996.

[3] M. C. Chan, Y.-J. Lin, and X. Wang, "A scalable monitoring approach for service level agreements validation," in *Proc. 8th Int. Conf. Network Protocol*, Osaka, Japan, Nov. 2000.

[4] T. Chen, S. Liu, M. J. Procanik, D. Wang, and D. Casey, "Inquire: A software approach to monitoring QoS in ATM networks," *IEEE Network*, pp. 32–37, Mar./Apr. 1998.

[5] P. Dini, G. V. Bochmann, T. Koch, and B. Kramer, "Agent based management of distributed systems with variable polling frequency policies," in *Proc. Integrated Network Management V*, San Diego, CA, May 1997. IFIP, pp. 553–564.

[6] A. Elwalid and D. Mitra, "Design of generalized processor sharing schedulers which statistically muliiplex heterogeneous QoS classes," in *Proc. IEEE INFOCOM*, New York, Mar. 1999, pp. 1220–1230.

[7] M. Grossglauser and D. N. C. Tse, "A time-scale decomposition approach to measurement-based admission control," in *Proc. IEEE INFOCOM*, New York, Mar. 1999, pp. 1539–1547.

[8] J. Jiao, S. Naqvi, D. Raz, and B. Sugla, "Minimizing the monitoring cost in network managemment," in *Proc. Integrated Network Management VI*, Boston, MA, May 1999. IFIP, pp. 155–170.

[9] S. Mazumdar and A. Lazar, "Objective-driven monitoring for broadband networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 8, June 1996.

[10] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The multiple node case," *IEEE/ACM Trans. Networking*, vol. 2, pp. 137–150, Apr. 1994.

[11] V. Paxson, "End-to-end internet packet dynamics," in *Proc. ACM SIGCOMM'97*, Cannes, France, Sept. 1997. ACM, pp. 139–152.

[12] V. Paxson, G. Almes, J. Mahdavi, and M. Mathis, "Framework for ip performance metrics," Internet Engineering Task Force (IETF), RFC 2330, May 1998.

[13] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," Internet Engineering Task Force (IETF), RFC 3031, Jan. 2001.

[14] S. Waldbusser, "Remote network monitoring management information base," Internet Engineering Task Force (IETF), RFC 1757, Feb. 1995.

[15] ——, "Remoate network monitoring management information base version 2 using SMIV2," Internet Engineering Task Force (IETF), RFC 2021, Jan. 1997.

[16] K. White, "Definitions of managed objects for service level agreements performance monitoring," Internet Engineering Task Force (IETF), RFC 2758, Feb. 2000.

[17] K. Yoshihara, K. Sugiyama, H. Horiuchi, and S. Obana, "Dynamic polling scheme based on time variation of network management infomation values," in *Proc. Integrated Network Management VI*, Boston, MA, May 1999. IFIP, pp. 141–154.

Yow-Jian Lin (M'88) received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, the M.S. degree in electrical and computer engineering from the University of California, Santa Barbara, and the Ph.D. degree in computer sciences from the University of Texas, Austin.

He is a member of the Technical Staff at Bell Labs Research, Lucent Technologies, Holmdel, NJ. From 1988 to 1995, he was with the Applied Research Area of Bell Communications Research (Bellcore), Morristown, NJ, first as a Member of the Technical Staff, then as a Director of Systems Specification and Management Research Group. He was a member of the Bellcore team that pioneered the feature interaction management research. He has been with Bell Labs Research since 1995. His current research interests include network monitoring and control, software systems management, feature modeling, and interaction detection.

Mun Choon Chan (M'97) received the B.S. degree from Purdue University, West Lafayette, IN, in 1990 and the M.S. and Ph.D. degrees from Columbia University, NY, in 1993 and 1997, respectively, all in electrical engineering.

From 1991 to 1997, he was a member of the COMET Research Group, Columbia University, working on ATM control and management. Since 1997, he has been a member of the Technical Staff at Bell Laboratories, Lucent Technologies, Holmdel, NJ. His current research include wireless data networking and network management.

Dr. Chan is a Member of the ACM and IEEE, and he serves on the Technical Program Committee of IEEE INFOCOM 2000–2002.