QoS protocol and algorithm join forces

Kuipers, FA

DOI
10.1109/CHINACOM.2006.344729

Publication date
2006

Document Version
Accepted author manuscript

Published in
Chinacom 2006; First international conference on communications and networking in chin

QoS protocol and algorithm join forces

F.A. Kuipers
Delft University of Technology, P.O. Box 5031, 2600 GA, Delft, The Netherlands.
F.A.Kuipers@ewi.tudelft.nl

***Abstract*— In general, routing is subdivided into two functionalities: routing protocols that keep the network state information up to date and routing algorithms that compute paths based on the information provided by the routing protocols. An excessive overhead prevents the distribution of all resource changes across the network, leading to stale link-state information. Besides inaccurate information, researchers have also resorted to inexact algorithms, because the general QoS path selection problem is NP-hard. Researchers have strived to optimize on either the protocol or the algorithm, but have hardly investigated how the two can benefit from eachother.**

In this paper we combine QoS protocol and algorithm. Our solution guarantees a tunable bounded error margin on the link-state information and provides an exact polynomial-time algorithm for the selection of (multi-constrained) paths.

I. Introduction

A network consists of nodes (routers and switches) and links (fiber, UTP, ...) and can be represented by a topology. In best-effort routing only knowledge about this topology is required. When specific quality requirements or constraints need to be obeyed, the topology information alone is not enough and also network resources need to be taken into account. Each link in the network is characterized by a set of measures, such as bandwidth, delay, jitter, packet loss, etc. Contrary to the topology information, these measures are highly volatile and it is difficult to maintain an accurate view of the state of these measures. Provided such information is available, one of the key issues in providing guaranteed Quality of Service (QoS) is *how to determine paths that satisfy QoS constraints*. The research community has extensively studied this problem, resulting in many QoS routing algorithms (see [5] for an overview and performance evaluation). Nevertheless, the accuracy of any algorithm depends on the accuracy of the network information. A Link-State Update (LSU) policy has the task of timely distributing network-state information, such that the algorithms can properly compute their paths. In this paper, we focus on a new type of LSU policy, referred to as ε-approximation LSU policy, which realizes ε-approximate link-state information, and we combine it with exact polynomial-time path selection algorithms.

We first introduce some terminology and notation. Let $G(\mathcal{N}, \mathcal{L})$ denote a network topology, where $\mathcal{N}$ is the set of N nodes and $\mathcal{L}$ is the set of L links. The number of QoS measures is denoted by m. Each link is characterized by an m-dimensional link-weight vector, consisting of m non-negative QoS weights ($w_i(u,v)$, $i = 1, ..., m$, $(u,v) \in \mathcal{L}$) as components. The QoS measure of a path can be either *additive* (e.g., delay), in which case the weight of a path equals the sum of the weights of its links, or min/max (e.g., available bandwidth), in which case the weight of a path is the minimum (or maximum) of the weights of its links. The m user requested QoS constraints are denoted by Q_i, $i = 1, ..., m$. The multi-constrained path selection problem in QoS routing can now formally be defined as follows:

Definition 1: Multi-Constrained Path (MCP) problem: Consider a network $G(\mathcal{N}, \mathcal{L})$. Each link $(u,v) \in \mathcal{L}$ is specified by a link-weight vector with as components m additive QoS link weights $w_i(u,v) \geq 0$. Given m constraints Q_i, the problem is to find a path P from a source node s to a destination node t such that

$$w_i(P) \stackrel{def}{=} \sum_{(u,v) \in P} w_i(u,v) \leq Q_i \qquad (1)$$

for all $1 \leq i \leq m$.

A path that satisfies all m constraints is referred to as a feasible path. Unfortunately, the MCP problem is NP-complete. To cope with this worst-case intractability, heuristics and ε-approximation algorithms have been proposed, as well as a few exact algorithms.

The main objective of this paper is to propose a QoS routing architecture consisting of both an LSU policy (part of a routing protocol) and a QoS algorithm. Due to the inherent overhead, it is virtually impossible to maintain exact information regarding the network state. The QoS algorithm itself, when not exact, may also be inaccurate; so there are two possible sources of inaccuracy. Exact QoS algorithms, e.g. SAMCRA [11], exist, which eliminate one of the sources of inaccuracy. However, their running-time is exponential in the worst case. On the other hand, ε-approximation algorithms run in polynomial time at the expense of a bounded ε-deviation from optimal. This means that the value of the returned solution is guaranteed to be within a factor $(1+\varepsilon)$ of the optimal value. In this paper, we combine an LSU policy and a QoS routing algorithm in such a way that the amount of inaccuracy in network-state information is bounded and tunable, and that the algorithm, which uses this information, is *exact* and runs in *polynomial* time.

We assume that the network-state information is flooded instantaneously across the network and that the admitted QoS flows are reserved.

The outline of the rest of this paper is as follows. In Section II the existing work on link-state update policies is sketched. Section III presents an ε-approximation link-state update policy that only takes into account the available bandwidth on links. This policy is evaluated via simulations in Section IV. Section V presents an overview of several ε-approximation techniques for path selection and Section

VI proposes a complete QoS architecture with bounded and tunable error. An evaluation of this approach is given in Section VII. Section VIII concludes the paper.

II. Related work

Key questions that have been investigated are *how to disseminate link-state information* and *when to disseminate link-state information*. The current Internet disseminates its network state through the entire autonomous domain by using flooding. In flooding, every router replicates the network-state information onto all of its outgoing links. This method is too costly when the frequency of updates is expected to be high. To reduce the overhead in flooding, tree-based protocols have been proposed [3]. The question when to update seems more difficult to answer. The proposed link-state update policies can roughly be classified into periodic or trigger-based LSU policies.

Periodic LSU policies periodically (with fixed predetermined intervals) update and disseminate the link-state information, regardless of the traffic load. Indeed, the smaller the update interval, the more updates and hence the more accurate the link-state information.

Periodic LSU policies are quite inflexible, because they do not take into account any "significant" changes in link state. Trigger-based policies, on the other hand, only distribute updates if a significant change (trigger) in the link state has occurred. The difficulty here is to determine what constitutes a significant change. Several trigger-based variations have been proposed:

- Class-based LSU policies: The total available bandwidth is classified into several classes. An update is triggered whenever the available bandwidth crosses from one class into another. A problem with class-based policies is that many updates may be triggered if the available bandwidth fluctuates around a class boundary.
- Threshold LSU policies: If the actual available bandwidth differs more than a certain threshold from the last advertised bandwidth, then an update is triggered. Let B^l be the last advertised available bandwidth and B^c the current available bandwidth, then an update is triggered when $|B^c - B^l| > \theta B^l$, where θ represents the threshold value.
- Hybrid LSU policies: Combinations of the above-mentioned policies have been proposed (e.g., [4]).
- Additional techniques: A hold-down timer is used to assure a minimum amount of time between two consecutive updates. A second technique, the moving average, does not look at the actual available bandwidth changes, but keeps track of the changes in the average bandwidth.

Because these policies do not distribute all changes, there inherently remains inaccurate information in the network. This resulted in the proposal of many QoS schemes that deal in some way with this inaccurate link-state information. However, only few have provided solutions with a quantifiable accuracy. Guérin and Orda [2] have proposed safety-based routing. Given the details of a trigger-based policy, safety-based routing computes the upper and lower bounds for the actual available bandwidth, based on the last advertised bandwidth. The inaccuracy is bounded and hence quantifiable. However, they complicate their approach by assigning a distribution function in between the upper and lower bounds to indicate the probability that a requested bandwidth is indeed available. In this paper, we avoid the determination of a probability distribution function and content ourselves with providing a quantifiable and tunable (in)accuracy in link-state information.

III. Bandwidth ε-approximation LSU

In this section we shall propose an ε-approximation LSU policy that only advertises the available bandwidth on a link. Exact polynomial-time algorithms exist that can compute the path with maximum available bandwidth. A popular algorithm that falls into this category is the Shortest-Widest Path algorithm [8].

We will focus on maximizing the available bandwidth. It is practically impossible to distribute all changes in available bandwidth, because that would create an unacceptably high overhead, which would cause a decrease in available bandwidth. Fortunately, not all changes are significant. We may consider changes that remain within a factor $(1 + \varepsilon)$ of the last advertised bandwidth to be insignificant. This can be achieved via the threshold-based link-state update policy. Let B^l be the last advertised available bandwidth and B^c the current available bandwidth, then an update is triggered when $|B^c - B^l| > \varepsilon B^l$, where ε corresponds to the threshold value. In case of a threshold-based LSU policy, an update is triggered unless $(1 - \varepsilon)B^l \leq B^c \leq (1 + \varepsilon)B^l$.

The formal definition of ε-approximation solutions requires that the cost of the returned path P is within a factor $(1+\varepsilon)$ of the *optimal* solution P^*. Optimal in this case refers to the path with minimal cost. When *maximizing* available bandwidth, we define that the available bandwidth of a path P should obey $|B^c(P^*) - B^c(P)| \leq \varepsilon B^c(P^*)$ in order for P to be an ε-approximation solution. Since bandwidth is a min/max parameter, for the current available bandwidth $B^c(P)$ of any path P holds $B^c(P) = \min_{(u,v)\in P}(B^c(u, v))$. Consequently, the error of a path P is determined by the error of the link on that path with the smallest available bandwidth. If the triggering rule is set to $|B^c - B^l| > \frac{\varepsilon}{2}B^c$, the advertised value B^l of any link (and hence path), which is used by the path selection algorithm, obeys $(1 - \frac{\varepsilon}{2})B^c \leq B^l \leq (1 + \frac{\varepsilon}{2})B^c$. Consequently, the advertised bandwidth of a path P can have an error of $\frac{\varepsilon}{2}B^c(P) \leq \frac{\varepsilon}{2}B^c(P^*)$, while the advertised error of the optimal path can have an error of $\frac{\varepsilon}{2}B^c(P^*)$. Both errors accumulate to a total error of $\varepsilon B^c(P^*)$ in the worst case, as required by our definition of ε-approximation.

IV. Simulations

In this section we present some results for the bandwidth ε-approximation LSU policy. The simulations were performed via a flow-level network simulator[1]. We used the MCI topology, with $L = 32$ links, displayed in Figure 1, which consists

[1]We have used DESINE, DElft SImulator of NEtworks, developed at Delft University of Technology.

of 11 edge and 8 core nodes. The links all were assigned a capacity of $C = 200$ units of bandwidth (e.g., Mb/s).

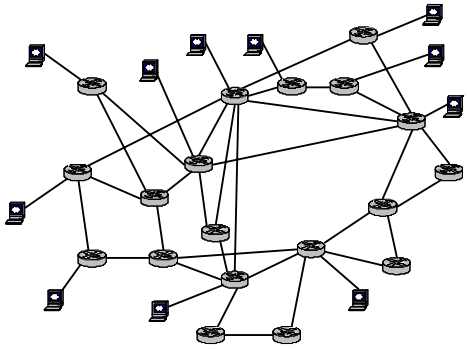

Fig. 1. The MCI topology, with $N = 19$ nodes and $L = 32$ links, which represents a realistic ISP topology. The (edge) nodes to which a computer is attached, receive flow requests. The remaining nodes are core nodes.

Each simulation iteration consisted of generating 11,000 flow requests, of which the first 1,000 were considered to be "warm-up" flows, which were not incorporated in the simulation results. Per simulation run five iterations were performed. The flow requests were generated according to a Poisson distribution, with inter-arrival time exponentially distributed with mean 1 (inter-arrival rate $\lambda = 1$), and flow duration exponentially distributed with mean $d = 50$. The source and destination nodes were chosen randomly. The mean number of hops h between any two nodes in the MCI topology is $h = 3.33$. The requested bandwidth of each flow was uniformly distributed between 1 and 10 with probability 0.7 and uniformly distributed between 80 and 100 with probability 0.3. Hence, the mean requested bandwidth b equals $b = 30.85$. According to [10], the network load ρ can be computed as $\rho = \frac{\lambda dbh}{LC} = 0.8$. We have simulated with the Shortest-Widest Path (SWP) algorithm and the Widest-Shortest Path (WSP) algorithm. The results are shown in Figure 2.

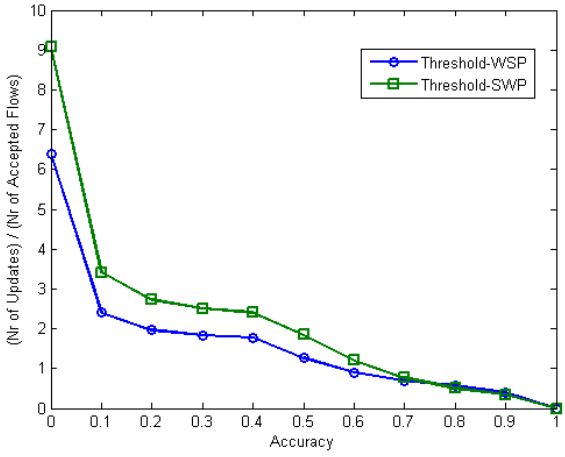

Fig. 2. Accuracy $\left(\frac{\varepsilon}{2}\right)$ versus overhead (per accepted flow) for the Widest-Shortest Path algorithm and the Shortest-Widest Path algorithm.

As expected, the overhead increases with an improving accuracy. The results also show that the WSP algorithm outperforms the SWP algorithm. Since WSP prefers minimum-hop paths, less links are affected by the allocation of paths and hence less updates are triggered.

V. ε-APPROXIMATION TECHNIQUES

The Simple Efficient Approximation (SEA [7]) algorithm is an ε-approximation algorithm that specifically targets the Restricted Shortest Path (RSP) problem [5]. SEA has one of the current best time complexities. An RSP algorithm is said to be ε-optimal if it returns a path whose cost is at most (1+ε) times the optimal value, where $\varepsilon > 0$ and the delay constraint is strictly obeyed. ε-approximation algorithms perform better in minimizing the cost of a returned feasible path as ε goes to zero. However, the computational complexity is proportional to $1/\varepsilon$, making these algorithms impractical for very small values of ε. SEA initially determines an upper bound (UB) and a lower bound (LB) on the optimal cost. Once suitable bounds are found, the approximation algorithm bounds the cost of each link by rounding and scaling it according to: $\tilde{c}(u,v) = \left\lfloor \frac{c(u,v)(N+1)}{\varepsilon LB} \right\rfloor + 1 \; \forall \; (u,v) \in \mathcal{L}$. Finally, it applies a pseudo-polynomial-time algorithm on these modified weights. SEA obtains the fully polynomial complexity of $O(LN(\log\log N + \frac{1}{\varepsilon}))$.

Besides rounding and scaling, two other ε-approximation techniques are presented in [9], namely interval partitioning and separation. With interval partitioning each node maintains a number of buckets/queues in which a single (sub)path is maintained. The number of buckets must be polynomial in the input size and ε to assure a polynomial time complexity. For instance, consider the MCP problem with $m = 2$. We choose to maintain $\frac{N}{\varepsilon}$ buckets per node, with the size of a bucket equal to $\frac{Q_2\varepsilon}{N}$. The buckets together therefore cover the range $(0, Q_2]$. A path P is stored in bucket i if $\frac{(i-1)\varepsilon Q_2}{N} < w_2(P) \leq \frac{i\varepsilon Q_2}{N}$ and $w_1(P) < w_1(P'),$ where P' is a previously stored path in bucket i. The total error that can be accumulated on a path is upper bounded by $\frac{NQ_2\varepsilon}{N} = Q_2\varepsilon$. Since we minimize on w_1, if $w_1(P) \leq (1+\varepsilon)Q_1$ we will have found our solution.

The separation technique only maintains paths (at a node) outside a certain distance of each other. For instance, a path P will only be stored if $w_i(P) - \frac{\varepsilon Q_i}{2N} \leq w_i(P') \leq w_i(P) + \frac{\varepsilon Q_i}{2N}$, for $i = 1, ..., m$ and $l(P) < l(P')$, where $l(P)$ refers to the length of path P.

VI. ε-APPROXIMATION LSU FOR MCP

Like bandwidth, the delay (and other QoS measures) on a link may also change frequently. Our goal is to develop an update policy that provides ε-accurate link state information and to combine this with a QoS algorithm that exactly solves the QoS path selection problem in polynomial time. We assume that the link weights are upper bounded and that this upper bound is known. Our assumption is valid in practice, where link weights are upper bounded and even have a finite granularity.

As we have seen in the previous section, ε-approximation algorithms may round and scale the cost of each link to ease the problem such that it becomes polynomially solvable. If only rounded and scaled weights are advertised, the routing

algorithm need not perform this task and can guarantee exactness in polynomial time. Another possible approach, based on interval partitioning, is to divide the link-weight spectrum into equal classes. We apply this to the MCP problem with $m = 2$ weights. Given the maximum link weights $w_i^{\max}$, then the advertised link weights may only take values in the range $\left[\frac{w_i^{\max}\varepsilon}{N}, w_i^{\max}\right]$, which are multiples of $\frac{w_i^{\max}\varepsilon}{N}$. Hence the continuous link cost spectrum is reduced to $\frac{N}{\varepsilon}$ distinct values/classes. The maximum value of a class is advertised by the LSU policy, when the actual weight crosses a boundary into a new class. The maximum link error that can be made equals $\frac{w_i^{\max}\varepsilon}{N}$, and hence the error along a path is upper bounded by $\varepsilon w_i^{\max}$. Since $w_i^{\max}$ is known, this is a quantifiable error margin[2]. At each node the algorithm needs to maintain $\frac{N \cdot N}{\varepsilon}$ buckets, because the total path weights are upper bounded by $N w_i^{\max}$. Note that our intention here is merely to indicate that it is possible to device link-state update policies with quantifiable (in)accuracy combined with exact polynomial-time algorithms. Improving upon the polynomial-time complexity of such algorithms is left for future work. We present our algorithm in Figure 3.

```
1. for all v ∈ N\s and all buckets x
2.       W1(v,x) ← ∞
3.       W2(v,x) ← ∞
4. W1(s,1) ← 0
5. W2(s,1) ← 0
6. queue H ← ∅
7. bucket x ← 1
8. INSERT(H, s, x, W1(s,x), W2(s,x))
9. while(H ≠ ∅)
10.      EXTRACT-MIN(H) → node i, bucket x
11.      if(i = t)
12.           if W1(i,x) ≤ Q1 + εw1^max
13.                return path
14.           else stop
15.      for each neighbor a of i
16.           if W2(i,x) + w2(i,a) ≤ Q2 + εw2^max
17.                find bucket y at a that fits
W2(i,x) + w2(i,a)
18.                if(W1(a,y) > W1(i,x) + w1(i,a))
19.                     W1(a,y) = W1(i,x) + w1(i,a)
20.                     W2(a,y) = W2(i,x) + w2(i,a)
21.                     DECREASE-
KEY(H, a, y, W1(a,y))
```

Fig. 3. Algorithm for the MCP problem, with $m = 2$.

Lines 1-5 initialize $W_i(v,x)$, where $W_i(v,x)$ corresponds to the i-th weight of the path from source s to node v, which is stored in bucket x. Each bucket can only contain one path. We make use of a heap structure H, e.g. Fibonacci or Relaxed heaps, to facilitate extracting, inserting, and decreasing paths. For more information on heap structures and the subfunctions EXTRACT-MIN, INSERT, and DECREASE-KEY, we refer to Cormen et al. [1].

The main algorithm starts at line 9. As long as H is not empty, we extract the path for which $W_1(i,x)$ is smallest over all nodes and buckets. If i equals the destination node t, we need to check if the constraint is not exceeded with more than $\varepsilon w_1^{\max}$, before returning the path. If the destination is not yet reached, lines 15-21 check for each neighbor a of i (except the previous node where the path came from), whether we can decrease $W_1(a,y)$ of the bucket y to which the new path belongs. Note that if bucket y is empty, we have to insert a path instead of decreasing its weight.

The extension of our algorithm to $m > 2$ is possible by increasing the number of buckets.

VII. SIMULATIONS

In this section we present some results for the MCP (with $m = 2$) ε-approximation LSU policy and algorithm (as displayed in Figure 3). The simulations setting was similar as described in Section IV. Again we have used DESINE with the MCI topology and the same parameters that led to a network load of 0.8. However in this case we also needed to define two link-weight functions, w_1 and w_2. Hence, each link is represented by two nodes, a capacity (or available bandwidth), and two link weights. A flow request constitutes of five elements: the source identifier, the destination identifier, a bandwidth constraint, and two QoS constraints. Based on these elements the path selection algorithm searches for a suitable path that obeys the constraints. If one is found, the path is set up. When a flow is allocated, the available bandwidth of the links along that path are reduced with the requested bandwidth (constraint). The link weights will also be affected. When a flow expires, the process is reversed. In our first simulation scenario we have adopted the following functions:

$$w_1(u,v) = C(u,v) - B_{av}(u,v), \quad w_2(u,v) = \frac{B_{av}(u,v)}{C(u,v)}$$

where $C(u,v)$ represents the capacity of link (u,v) and $B_{av}(u,v)$ the available bandwidth. In the second scenario the two link weights were correlated to each other with correlation coefficient ρ. In all scenarios the QoS constraints were chosen uniformly from the range $[1, Q]$ and the maximum link weights were set to $w_i^{\max} = 200$, for $i = 1, 2$. When one of the link weights triggers an update, immediately the other parameters are also updated. Figures 4 and 5 present the results for our first scenario.

Figure 4 reveals that with increasing ε, more flows are accepted, because the constraints become more relaxed. This increase is more pronounced when the constraints are strict than when they are loose, which is clearly seen by the slope of the three lines.

Hence, two factors will affect the number of updates. If ε is large the granularity in link weights is coarser and consequently less updates are necessary. On the other hand, a larger ε also results in more accepted flows, which will trigger more updates. The decrease in the number of updates per accepted flow in Figure 5 demonstrates that the coarser granularity outweighs the increase in accepted flows in terms of generated

[2]Formally, it does not obey the strict definition of ε-optimality for MCP, since it may occur that $\varepsilon w_i^{\max} > \varepsilon Q_i$.

overhead. We can also observe that the number of updates is higher than was the case for only one parameter (bandwidth as studied in Section IV), because now two parameters can trigger an update.

Figure 6 present the overhead for the second scenario, in which the two link weights are correlated.

Also in this case an increase of ε resulted in an increase of the number of accepted flows and a decrease in the overhead caused by link-state updates. The correlation coefficient did not have a big influence on the acceptance ratio. With a negative correlation, more updates are expected, because when the first link weight is small and does not trigger an update, the second link weight is likely to be large and will cause an update.

Finally we want to remark that, similar to [6], in all simulated instances the actual deviation from optimal was much smaller than the maximum theoretical factor $(1+\varepsilon)$.

VIII. Conclusions

In this paper we have presented a QoS solution to device QoS algorithms that take into account the properties of the link-state update (LSU) policy. When these two entities are considered separately, we introduce two sources of inaccuracy (in LSU policy and algorithm) or require exponential-time exact algorithms. Our solution eliminates the inaccuracy in the algorithm and thereby guarantees a polynomial-time complexity, whereas the inaccuracy in the LSU policy is bounded and controlled by the ISP.

IX. Acknowledgements

The help of A. Beshir and B. Fu with the simulations, and the comments of P. Van Mieghem have been greatly appreciated.

References

[1] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *An Introduction to Algorithms*, MIT Press, Boston, 2000.

[2] R. Guerin and A. Orda, "QoS routing in networks with inaccurate information: theory and algorithms," IEEE/ACM Transactions on Networking, vol. 7, no. 3, pp. 350-364, June 1999.

[3] Y. Huang and P.K. McKinley, "Tree-based link-state routing in the presence of routing information corruption," Computer Communications, vol. 26, pp. 691-699, 2003.

[4] M. Kabatepe and M.G. Hluchyj, "On the effectiveness of topology update mechanisms for ATM networks," Proc. of ICC'98, no. 1, pp. 1134-1139, June 1998.

[5] F.A. Kuipers, T. Korkmaz, M. Krunz and P. Van Mieghem, "Performance Evaluation of Constraint-Based Path Selection Algorithms," IEEE Network, vol. 18, no. 5, pp. 16-23, September/October 2004.

[6] F.A. Kuipers, A. Orda, D. Raz, and P. Van Mieghem, "A comparison of exact and ε-approximation algorithms for constrained routing," Proc. of Networking'06, Coimbra, Portugal, May 15-19, 2006.

[7] D.H. Lorenz and D. Raz, "A simple efficient approximation scheme for the restricted shortest path problem," Operations Research Letter, vol. 28, no. 5, pp. 213-219, June 2001.

[8] Q. Ma and P. Steenkiste, "Quality-of-Service routing for traffic with performance guarantees," Proc. of IWQoS'97, Columbia University, New York, May 1997.

[9] S. Sahni, "General Techniques for Combinatorial Approximation," Operations Research, vol. 25, no. 6, pp. 920-936, 1977.

[10] A. Shaikh, J. Rexford and K.G. Shin, "Evaluating the Impact of Stale Link State on Quality-of-Service Routing," IEEE/ACM Transactions on Networking, vol. 9, no. 2, April 2001.

[11] P. Van Mieghem and F.A. Kuipers, "Concepts of exact quality of service algorithms," IEEE/ACM Transactions on Networking, vol. 12, no. 5, pp. 851-864, October 2004.

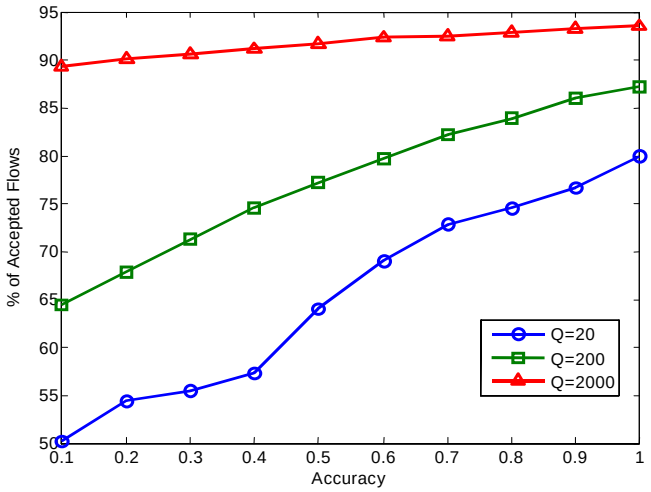

Fig. 4. Percentage of the accepted flows over the total number of flow requests as function of the accuracy ε.

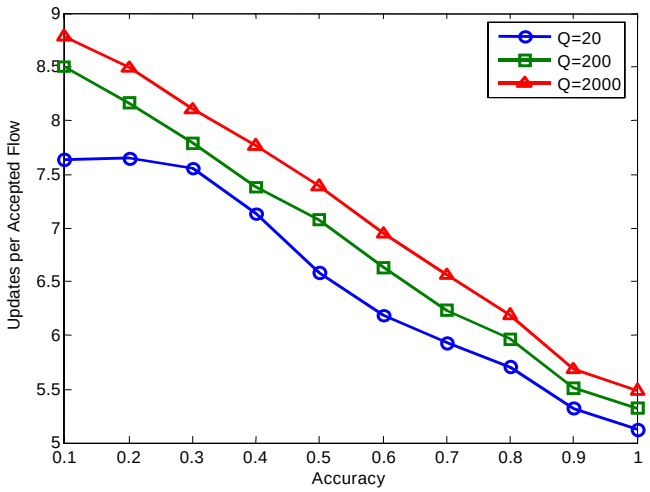

Fig. 5. Expected number of updates that were triggered per accepted flow, as function of the accuracy ε.

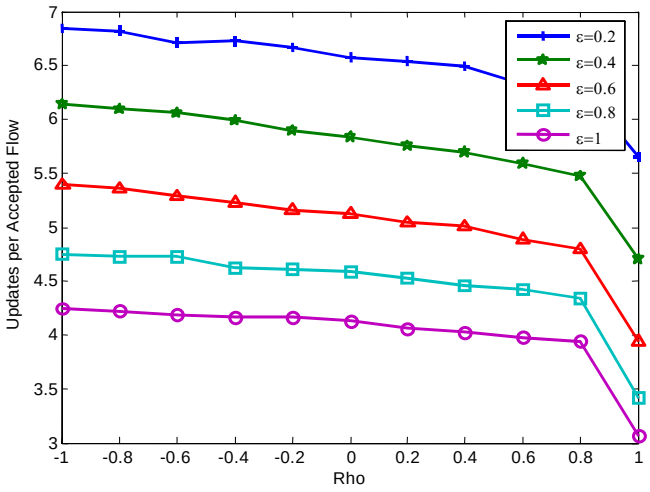

Fig. 6. Expected number of updates that were triggered per accepted flow, for different ε, as function of ρ. $Q = 1000$.