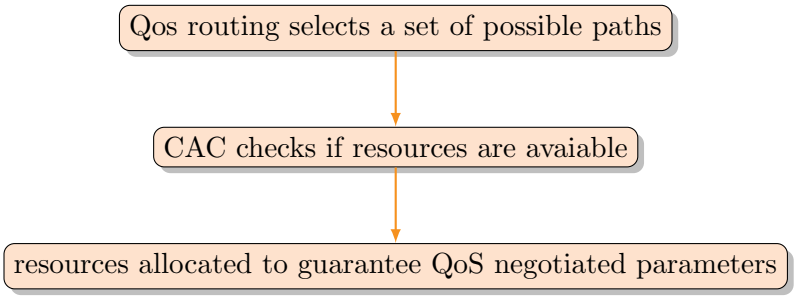
QoS Routing and CAC

Introduction

Mainly it is a preventive traffic control technique, but can become reactive with renegotiation. In order to accept a call:



Possible paths selected by the initial step are paths over which is possible to run CAC algorithms: in this way, reported in the second step, those algorithms checks if QoS parameters are respected and, if they are, resources are allocated.

This procedure can be applied both to unicast and multicast calls, but, at first, the concept of *call* have to be defined; for example in ATM a call is each VPI/VCI, in Frame Relay is each DLCI while in Internet it is more difficult since service is datagram so flow identification it is not trivial.

Multicast is very difficult to treat: one possible solution is an implementation in which κ independent and unicast flows are sent by the source. It means that the source have to be able to generate those flows and, anyway, it is a bad manner of managing the issue. Another solution is generate a single flow and duplicate it at the last point as possible: it is a better approach than the previous one, but there are at least two issues. The first is: how to find the point in which duplication have to performed? Notice that the node is supposed to be able to generate κ flows. The other issue is that scalability is not realized. Moreover, there are two facts that have to be taken into account:

. the number of users may change in time, so the way in which users join the multicast service have to be managed;

. user's requirement are very heterogeneous so they need different bandwidth.

QoS routing

In general QoS routing is very similar to *traditional* routing; the network can be modeled as a graph in which:

. *nodes* are switches and routers;

. *edges* are links.

The traditional routing problem have to find the path with the minimum cost among all path that connect two users; the way in which costs are associated to edges is very important: infact, if all of them are equal to 1, the path with the minimum cost is exactly the shortest path. It means that the minimum amount of resources is consumed so the number of admissible calls is maximize. Instead if costs are different this assumption is not true: the shortest path can not present the minimum cost.

The QoS routing forecast that the call required of a users have some QoS requirements: nodes may have a state related to QoS metrics and also edges so routing is simply the operation that find over the graph a path which respects all requirements. It can be possible that the path found it is not the one with the minimum cost. The principal problem is the diversity of QoS requirements; they can be:

. bit rate, delay, delay jitter, loss ratio; they can be *additive*, *multiplicative*, *concave constraints*;

. multiple constraint can make QoS routing problem NP.

Another problem is the integration with best-effort traffic that is always accepted, but since QoS traffic has higher priority, if needed best-effort traffic is not consider. The principal issue, instead, is that network state change in time: it means that costs change in time so:

. measures have to be taken quickly;

. information has to be propagated to all nodes;

. a new path computation have to be performed.

Notice that, if states information is outdated, wrong paths are computed so performances degrade significantly while if all goes right the algorithm is very reactive; in both cases the overhead is relevant.

State information associable to each node (link state) usually is a triple composed by:

. bandwidth;

. delay;

. cost.

Node state is simply a combination of its own link state but the processing capability have to be taken into account; this feature takes care of the link speed and the number of packets: if the packet size is small then CPU resources are consumed a lot while if packets are huge few resources are consumed.

Information is exchanged by using *link-state* protocols or *distance-vector* protocols and scalability is obtained by aggregate information.

Unicast (Multicast) QoS Routing

Given:

. a source node s;

. a destination node d (for multicast a set of destination D);

. a set of QoS constraint C;

the routing operation try simply to find the best path from s to d (or D) which satisfies C.

Unicast classification There are 4 possible kinds of classes:

. link-optimization (LO);

. link-constrained (LC);

. path-optimization (PO);

. path-constrained (PC).

Link kinds define the state of a path looking at the bottleneck link and maximize the residual bandwitdth; for example:

. in a link-optimization scenario the objective is find a path with the largest bandwidth on a bottleneck link;

. in a link-constrained scenario the objective is find a path whose bottleneck link is above a given value.

Notice that a link-constrained problem can be solved with a link-optimization approach.

Path kinds define the state of the path determining the combined state over all links of the path, considering, as metric the delay; the two cases are:

. in a path-optimization scenario the objective is find the path with the minimum total cost (in terms of delay);

. in a path-constrained scenario the objective is find the path whose metric (delay) is bounded by a given value.

Elementary routing problems seen previously can be combined in different ways; an example is LC-PO problem: bandwidth constrained least delay routing. In general the following combinations can be solved in a polynomial time:

. LO-LC;

. LC-PO;

. LC-PC;

. PC-LO.

Instead, the combination PC-PO, is usually an NP problem (find the least cost path with a bounded delay) because the two metrics are independent. If measures are not real numbers or unbounded integers the problem becomes polynomial time solvable.

Strategies The way in which information is maintained and how the feasible path is found defines the following strategy classification:

. source routing: centralized solution, avoid all problems due to distributed solutions (deadlock, loops, distributed termination), but there is a large communication overhead to update states which implies some imprecision of network status; moreover the overhead is very high and it is possible reduce it only reducing the number of information exchanged, but this means that routing is based on very old information state;

. distributed routing: this approach is more scalable but suffers of problems previously cited;

. hierarchical routing: it is often used in conjuction with source routing since it reduce the number of information required using aggregation techniques; this fact implies that some imprecision is added because the information on which routing is applied is partial.

Proposed algorithms

. widest path: maximize the avaiable bandwidth;

. shortest path: minimize the delay;

. shortest-widest path: among widest path, the shortest one is selected;

. widest-shortest path: among shortest path, the widest one is selected;

. delay constrained least-cost routing.

Examples of source routing algorithms:

. bandwidth-delay constrained: all links which have not enough bandwidth are not considered, then the shortest path is selected;

. transform delay, jitter and buffer space bounds in bandwidth bounds if traffic is token bucket controlled (WFQ provides bandwitdth guaranteed if traffic is delay bound).

Issues in unicast traffic For high loads the maximum throughput is provided by the minimum hop since it means that the minimum amount of resources are consumed for each call: in this way the number of avaiable call increases. If the scenario presents an high load with no minimum hop network performances will be statistically penalized. For medium and low loads performances should depend on network topology.

Some algorithms are only implemented in a centralized way so, since they do not have a global view, hop by hop decisions are sub-optimal just because are local.

Multicast classification It is similar to unicast case, but optimization or constraints must to be applied to the full tree and not on the path. The Steiner tree problem try to find the least-cost tree, but it is an NP problem if destinations set does not include all network nodes, otherwise the problem can be simply resolved because it is exactly the minimum spanning tree problem (solvable in polynomial time).

Issues in multicast traffic The first thing to keep in mind is that multicast trees are dynamical trees, in which users can join or leave in any time: the issue therefore is maintain updated the tree while calls are active.

Receivers are heterogeneous so the way in which resources are allocated is very important; if, for example, users are partitioned according to a given attribute, probably they get the same quality as the worse one. Another possible solution is divide them into different layers associated to a priority: users with low priority get less quality while users with an higher priority get more quality.

Connection admission control

CAC algorithms checks, among possible paths, those that does not violate QoS parameters and does not violate requirements of already set calls. Their execution is done in all network nodes on which calls are routed: if requirements are satisfied, the call is accepted, while in the other case it can be refused or a re-negotiation phase take place. CAC methods are principally:

- parameters based admission control:
  - peak/average rate;
  - worst case analysis;
  - equivalent bandwidth;
- measurements based admission control.

Peak rate

Peak rate allocation says that a call κ is accepted if the avaiable bandwidth is large than the peak bandwidth of call κ:

$$B_p^{(\kappa)} \leq C - \sum_i B_p^{(i)}$$

where:

- $B_p^{(\kappa)}$ is the bandwidth associated to call κ;
- C is the total capacity;
- $\sum_i B_p^{(i)}$ is the bandwidth associated to all previous accecpted calls.

This approach is a worst case approach and its behavior with CBR/VBR sources is:

- for CBR some considerations are similar to TDM scheme with circuit switching, but here synchronization is not present; moreover the delay is a function of accepted calls and, if the buffer size is proportional to that number, there are no losses;
- for VBR sources there are the same guarantees of CBR sources but the link utilization L_u is proportional to:

  $$L_u = \frac{B_m}{B_p}$$

  instead for CBR sources is equal to 1.

Considerations This algorithm is simple but it does not exploit benefits of statistical multiplexing; it guarantees very well QoS requirements, but link utilization can be largely under-utilized for VBR traffic (this is due to the ratio B_m over B_p).

If many links are crossed CBR sources may become VBR source if the shaper does not preserve the shape of traffic; among possible schedulers, the FIFO scheduler, is the one that is typically used, but if the delay constraint become tight or bandwidth is increased reducing the number of admissible calls or the type of scheduler is changed.

Average rate

With this approach a call κ is accepted if the avaiable bandwidth is large than the average bandwidth of call κ:

$$B_m^{(\kappa)} \leq C - \sum_i B_m^{(i)}$$

The objective is maintain the network status over long time never overloaded; of course the longness of time period have to be defined.

Considerations As the previous method, also this is a simple approach with maximize the link utilization; if the buffer has infinite size losses are null, but in real case congestion can occur and the link is overloaded proportionally to source burstiness. This can happend for a short time or not, anyway in this case the number of losses in uncontrolled as delays.

Example worst case analysis

This example is another case of worst case approach: suppose that the source is constrained by a token bucket. The call will be accepted if the two following conditions are both verified:

. the summation of token rates is smaller than the link capacity;

. the summation of token depth is less than avaiable buffer space.

The approach have these properties:

. no losses;

. low utilization;

. delay guarantees depending on:

  . number of already accepted calls;

. token depth.

If the scheduler used is WFQ, bandwidth can be allocated in order to satisfy the worst delay along the path and define a bound to the buffer size to avoid packet losses.

Equivalent bandwidth

Given:

. a specific traffic characterization in terms of peak rate, average rate, burst duration;

. some QoS requirements;

. traffic behavior of other calls;

the equivalent bandwidth is the bandwidth needed to satisfy call QoS requirements. With this approach a call κ is accepted if:

$$B_{eq}^{(\kappa)} \leq C - \sum_{i} B_{eq}^{(i)}$$

The more stricts are QoS requirements the more larger will be the equivalent bandwidth assigned, but how it is computed? A traffic model is needed in which:

. the source behavior have to be defined in a stochastic way;

. emulate or solve the system who comprises all already accepted calls plus the new one;

. define how much bit rate have to be assigned to the new call satisfing all QoS requirements.

In a theoretical point of view the best model is that one close to the realistic behavior of traffic, but it means that it is very complex and difficult to compute; otherwise less complex models are easy to implement but they are not so realistic.

Values taken by the equivalent bandwidth can be:

. for sure greater than the average rate;

. if delay constraints are very tight with overallocation the equivalent bandwidth can be greater than the peak rate;

. usually its value is between the average rate and the peak rate:

$$B_m \leq B_{eq} \leq B_p$$

Using equivalent bandwidth for CAC allows to compute a proper service rate adapt to guarantee call QoS, but it works good if the model adopted is realistic otherwise it is difficult to implement in a sequence of links because multiplexing changes the traffic shape. In that way computation will not be independent.

Another solution instead defining a model is to precompute a set of traffic classes: each one is identified by a given traffic characterization and given QoS requests. Users have to choose among possible sets that one which satisfy their needs: in this way computation can be done off-line (while in the other case all computation are done in real-time) and all complexity due to real-time changes is avoided. Infact, with already defined sets, the number of calls accaptable for each set is known a priori and when that number is reached the call is simply refused. Classes are defined based on all possible kind of application run by users so a big disadvantage is that if a new application is developed sets have to be recomputed, expecially if that application is very popular.

Perhaps the best solution is mixed the two previous mentioned, for example assigning a certain percentage to the *real-time* method and the remaining part to the *off-line* approach.

Measurement based CAC

This approach is much simpler than parameter based CAC and much real: it is not needed a macth between the model with the real case since, using measures, requirements needed are surely satisfy; moreover, notice that, measures are avaiable for free becasue network devices provide them. With the parameter method the user have to specify which traffic characterization needs, instead, collecting measures avoid this fact becasue the traffic characterization is determined by the network itself.

The basic idea is that in real time, for each link, measures are taken on the traffic load for a pre-defined measurement interval in order to compute exaclty the residual avaible bandwidth. The call κ is accepted if:

$$B_p^{(\kappa)} \leq B_{measured\ avaiable\ bit\ rate}$$

The important point to remark is that now the call is accepted on the base of its own real traffic and not on the base on given parameters. For example, in the previous formula, the peak rate approach is used to make the comparison, but from the avaible bandwidth it is subtract the real load of the call measured instead the peak rate.

This approach is very effective if the traffic characterization or newtork status are not known or are known with a large error; the network utilization is very high so, as a consequence, it is very difficult provide QoS.

Of course this method suffers classical issues due to measures, for example, measurement errors or how long is taken the window interval of time. If it is too large, as usual, the approach guarantees stability but it is less reactive than a window size smaller, but, using a reduced window size implies less stability with respect to the usual traffic behavior.

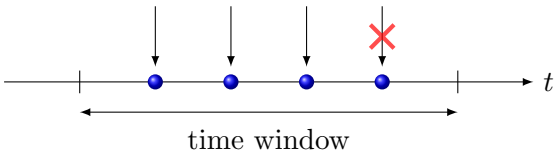
Moreover, an implicit assupmtion is considered: measures taken in a moment characterize the traffic behavior of the next time window; it means that traffic is supposed to be stable, but this can not be always true: it depends, also, on which application is considered. For example, traffic generated by a video is stable, while traffic generated by a data transfer application no. In order to have a more detalied situation it is possible to reduce the time window: it is reduced the traffic unpredictability, but it implies that measures are taken more frequently and the algorithm is more reactive.

Another issue is the management of the possibility that too calls arrive in a given time window: at least there are two approaches:

. conservative approach: during a time window peak rate allocation is considered; in this way it may happend that a call is refused while there is enough bandwidth checking the real traffic generated by accepted calls;

. non conservative approach: more calls are accepted, but it may happend that, taken measures in the next time window, the real load is exceeded; this approach is very dangerous with large window intervals.

Example Consider a case in which each call has a bandwidth in terms of peak rate of 10 Mbit and the residual avaible bandwidth is 30 Mbit.



With conservative approach the 4° call will be refuse since there is no enough bandwidth, but it is possible that the average rate of the 3 accepted calls takes only 17 Mbit: it means that the residual bandwidth is 13 Mbit so the refused call should have been accepted. Notice that, more larger is the time

window, the larger will be the probability of having more calls.

Another issue is the way in which measures are collected. If they are a lot, the management requires time and wastes processing resources, but, in order to take reliable decisions, information has to be known in a small time.

This approach is useful only for CAC, but is almost useless for QoS; infact, measures can be taken on already accepted calls, hoping that future behavior will be stable, but in order to guarantee QoS, parameters have to be known and set before. Considering the delay constraint a solution is adopt particular scheduler: for example, a work-conserving algorithm, with a data input capacity bounded allows to have a bounded delay.

CAC issues

The main problem is the unfairness among high bit rate calls and low bit rate calls in a saturated scenario: infact, with more probability low bit rate calls will be accepted so it may create a sort of starvation and it implies that the blocking probability, which is a QoS parameter, increases. A solution is partition the resources, giving a certain percentage of them to high bit rate calls and the remaining part to low bit rate calls. In this way the blocking probability is bounded. Another solution is partition the resources according to the current load: if the situation is a saturated scenario, low calls can be refuse, hoping that in the future an high bit rate call will be accepted.

Another issue is that is difficult to extend algorithms to several consecutives links, infact, the measurment approach works well since measures are taken independently for each link. In the other case, when statistical multiplexing is not adopted, the equivalent bandwidth may occurs relevant errors when computations are performed in the core of the network. Notice that, hop by hop approach, does not provide and end to end quality which is the one really relevant for users.