BANDWIDTH ALLOCATION PLANNING IN COMMUNICATION NETWORKS

Christian Frei and Boi Faltings
Artificial Intelligence Laboratory
Swiss Federal Institute of Technology (EPFL)
CH-1015 Lausanne, Switzerland
`Christian.Frei@epfl.ch`

Abstract

Increasing usage of communication networks and quality of service requirements call for efficient management of these high-speed networks. We address the problem of bandwidth allocation planning of communication demands. Shortest path routing is the traditional technique applied to this problem. However, this approach can lead to poor network utilization and even congestion. We show how an abstraction technique combined with systematic search algorithms and heuristics derived from Artificial Intelligence make it possible to solve this problem better and in much tighter networks, from a bandwidth usage point of view.

Keywords: QoS-routing, constraint-based routing, resource allocation planning.

1 Introduction

The communication networks of the next millennium are expected to offer a wide range of services to an increasingly large number of users, with a diverse range of Quality of Service (QoS) requirements. This calls for efficient control and management of these high-speed networks. A central problem is the automatic routing of traffic through the network. Currently, shortest path routing is most often used to route traffic across a network. Although this ensures the best possible route for each particular demand, it can lead to ineffective use of the global network and even congestion, especially in highly loaded networks. From the routing point of view, the key resource to manage in networks is bandwidth. Therefore, in order to make better use of available network resources, there is a need for planning the bandwidth allocation to communication demands, in order to set up routing tables (or any other route selection criterion) more purposefully. This can be achieved by the use of *global information*, including not only the available links capacities but also the expected traffic

profile, in an off-line manner. The traffic profile may be given, as when setting up virtual private networks in an ATM network of a provider, or estimated by objective traffic measurements.

Formally, we define the problem of resource allocation in networks (RAIN) as follows:

Given a network composed of nodes and links, each link with a given resource capacity, and a set of communication demands to allocate, each demand defined by a triple (*source*, *destination*, *requested bandwidth*),

Find one route for each demand so that the bandwidth requirements of the demands are simultaneously satisfied within the resource capacities of the links.

Because of performance reasons, the demands are not divided among multiple routes. With this restriction, the RAIN problem is NP-hard in the number of demands. When demands are subject to multiple additive or multiplicative quality of service (QoS) criteria, then [4] has shown that the allocation of every single demand is NP-hard by itself. This creates a new situation for the networking community, as traditional routing algorithms such as shortest paths do not perform very well on this problem.

Constraint satisfaction [3] is a technique which has been shown to work well for solving certain NP-hard problems. Indeed, the RAIN problem is easily formulated as a *Constraint Satisfaction Problem* (CSP): variables are demands, the domain of each variable is the set of all routes between the endpoints of the demand, and constraints on each link must ensure that the resource capacity is not exceeded by the demands routed through it. A solution is a set of routes, one for each demand, respecting the capacities of the links.

However, this formulation presents severe complexity problems. It is computationally too expensive to compute the domains of the variables, i.e., all the routes that join the endpoints of each demand. Suppose the

network is simple but complete (this is not even the worst case, since a communication network is a multi-graph: it allows multiple links between same endpoints) with n nodes. A route is a simple path, its length in number of links is therefore bounded by $n - 1$. Since a route of length j has $j - 1$ intermediate (and distinct) nodes, the number of routes of length j is $(n-2)!/(n-j-1)!$. The total number of routes between two nodes is therefore equal to $\sum_{i=1}^{n-1} (n-2)!/(n-i-1)!$. Storing all routes between a pair of nodes would require exponential space. For instance, in a complete graph with 10 nodes, there are 69'281 routes between any pair of nodes. Explicit representation of the domains is thereby impossible.

In this paper, we show how abstraction of the network creates a compact representation of the problem, which allows applying well-known CSP techniques such as forward checking, variable and value ordering to the RAIN problem, with manageable complexity.

There has been surprisingly little published research on the RAIN problem. Currently, most network providers use some kind of best effort algorithm, without any backtracking due to the complexity of the problem. Given an order of the demands, each demand is assigned the shortest possible route supporting it, or just skipped if there is no such route. Mann and Smith [2] search for routing strategies using genetic algorithms and simulated annealing approaches that attempt to ensure: that no link is over-utilized (hard constraint); if possible, that all links are evenly loaded (below a fixed target utilization), for the predicted traffic profile; and, that the routing assignment minimizes the communication costs. However, their methods do not apply well to highly loaded networks.

2 The Blocking Island Paradigm

[1] introduces a clustering scheme based on Blocking Islands (BI), which can be used to represent bandwidth availability at different levels of abstraction, as a basis for distributed problem solving. A β -blocking island (β -BI) for a node x is the set of all nodes of the network that can be reached from x using links with at least β available resources, including x . Figure 1 (d) shows all 64K-BIs for a network.

Blocking islands have some fundamental properties. β -BIs partition the network into equivalence classes of nodes. The BIs are *unique*, and *identify global bottlenecks*, that is, inter-blocking island links. If inter-BI links are links with low remaining resources, as some links inside BIs may be, inter-BI links are links for

which there is no alternative route with the desired resource requirement. Moreover, BIs highlight the *existence* and *location* of routes at a given bandwidth level:

Proposition 1: *There is at least one route satisfying the resource requirement of an unallocated demand $d_u = (x, y, \beta_u)$ if and only if its endpoints x and y are in the same β_u -BI. Furthermore, all links that could form part of such a route lie inside this blocking island.*

Blocking islands are used to build the β -Blocking Island Graph (β -BIG), a simple graph representing an *abstract* view of the available resources: each β -BI is clustered into a single node and there is an abstract link between two of these nodes if there is a link in the network joining them. Figure 1 (c) is the 64-BIG of the network of Fig. 1 (d). An abstract link between two BIs clusters all links that join the two BIs, and the abstract link's available resources is equal to the maximum of the available resources of the links it clusters (since a demand can only be allocated over one route). These abstract links denote the critical links, since their available resources do not suffice to support a demand requiring β resources.

In order to identify bottlenecks for different β s, e.g., for typical possible bandwidth requirements, we build a recursive decomposition of BIGs in decreasing order of the requirements: $\beta_1 > \beta_2 > \dots > \beta_b$. This layered structure of BIGs is a *Blocking Island Hierarchy* (BIH). The lowest level of the BIH is the β_1 -BIG of the network graph. The second layer is then the β_2 -BIG of the first level, i.e., β_1 -BIG, the third layer the β_3 -BIG of the second, and so on. On top of the hierarchy there is a 0-BIG abstracting the smallest resource requirement β_b . The abstract graph of this top layer is reduced to a single abstract node (the 0-BI), since the network graph is supposed connected. Figure 1 shows such a BIH. The graphical representation shows that each BIG is an abstraction of the BIG at the level just below (the next biggest resource requirement), and therefore for all lower layers (all larger resource requirements).

A β -BI for a node x of a network graph can be obtained by a simple greedy algorithm, with a linear complexity of $O(m)$, where m is the number of links. The construction and maintenance of a BIH can be carried out in $O(bm)$, where b is the number of different bandwidth requirements.

3 Routing as a CSP

Solving a RAIN problem amounts to solving the CSP introduced in Sect. 1. This can be done using a *backtracking algorithm* with *forward checking* (FC) [3]. Its

Fig. 1: The blocking island hierarchy for resource requirements $\{64K, 56K, 16K\}$. The weights on the links are their available bandwidth. Abstract nodes' description include only their node children and network node children in brackets. Link children (of BIs and abstract links) are omitted for clarity, and the 0-BI is not displayed since equal to N_7 . (a) the 16-BIG. (b) the 56K-BIG. (c) the 64K-BIG. (d) the network.

basic operation is to pick one variable (demand) at a time, assign it a value (route) of its domain that is compatible with the values of all instantiated variables so far, and propagate the effect of this assignment (using the constraints) to the future variables by removing any inconsistent values from their domain. If the domain of a future variable becomes empty, the current assignment is undone, the previous state of the domains is restored, and an alternative assignment, when available, is tried. If all possible instantiations fail, backtracking to the previous past variable occurs. FC proceeds in this fashion until a complete solution is found or all possible assignments have been tried unsuccessfully, in which case there is no solution to the problem.

The formulation of the CSP presents severe complexity problems (Sect. 1). Nonetheless, BIs provide an abstraction of the domain of each demand, since any route satisfying a demand lies within the β -BI of its endpoints, where β is the resource requirement of the demand (Proposition 1). Therefore, there is at least one route satisfying the demand if the endpoints of a demand are clustered in the same β -BI. We do not know what the domain of the variable is *explicitly*; however we do know it is non-empty.

Forward Checking Thanks to the route existence property, we know at any point in the search if it is still possible to allocate a demand, without having to compute a route: if the endpoints of the demand are clustered in the same β -BI, where β is the resource requirement of the demand, there is at least one, i.e., the domain of the variable (demand) is not empty. Therefore, after allocating a demand, forward checking is performed first by updating the BIH, and then by checking that the route existence property holds for all uninstantiated demands. If the latter does not hold, another route must be tried. Domain pruning is thus implicit while maintaining the BIH.

Variable Ordering A backtracking algorithm involves two types of choices: the next variable to assign, and the value to assign to it. The selection of the next variable to assign may have a non-negligible effect on search efficiency. A widely used variable ordering technique is based on the “fail-first” principle: “To succeed, try first where you are most likely to fail”. The idea is to minimize the size of the search tree and to ensure that any branch that does not lead to a solution is pruned as early as possible when choosing a variable. There are some natural static variable ordering (SVO) techniques for the RAIN problem, such as choose first the demand

that requires the most resources. Nonetheless, BIs allow dynamic (that is during search) approximation of the difficulty of allocating a demand in more subtle ways by using the BIH:

DVO-HL (Highest Level): first choose the demand whose lowest common father of its endpoints is the highest in the BIH (remember that high in the BIH means low in resources requirements). The intuition behind DVO-HL is that the higher the lowest common father of the demand’s endpoints is, the more constrained (in terms of number of routes) the demand is. Moreover, the higher the lowest common father, the more allocating the demand may restrict the routing of the remaining demands, since it will use resources on more critical links.

DVO-NL (Number of Levels): first choose the demand for which the difference in number of levels (in the BIH) between the lowest common father of its endpoints and its resources requirements is lowest. The justification of DVO-NL is similar to DVO-HL.

Value Ordering The domains of the demands are too big to be computed beforehand. Instead, we compute the routes as they are required. In order to reduce the search effort, routes should be generated in “most interesting” order, so to increase the efficiency of the search, that is: try to allocate the route that will less likely prevent the allocation of the remaining demands. A natural heuristic is to generate the routes in *shortest order* (SP), since the shorter the route, the fewer resources will be used to satisfy a demand. However, we can do better with a kind of min-conflict heuristic, based on the BIH, called *lowest level* (LL) heuristic. It considers first (in shortest order) the routes in the lowest BI (in the BIH), i.e., the BI for the highest resource requirement, clustering the endpoints of the demand. This heuristic is based on the following observation: the lower a BI is in the BIH, the less critical are the links clustered in the BI. By assigning a route in a lower BI, a better *bandwidth connectivity* preservation effect is achieved, therefore reducing the risk of future allocation failures. Bandwidth connectivity can therefore be viewed as a kind of *overall load-balancing*. Moreover, the lower a BI is, the smaller it is in terms of nodes and links, thus reducing even more the search space when looking for the first routes, and thereby achieving a computational gain during the early stages of the search. Generating one route with the LL heuristic can be done in linear time in the number of links.

4 Results and Conclusion

In practice, the RAIN problem poses itself in the following way: a service provider receives a request from the customer to allocate a number of demands, and must decide within a certain decision threshold (for example, 5 seconds), whether and how the demands could be accepted. A meaningful analysis of the performance of the heuristics we proposed would thus analyze the probability of finding a solution within the given time limit, and compare this with the performance that can be obtained using common methods of the networking world, in particular shortest-path algorithms.

For comparing the efficiency of different constraint solving heuristics, it is useful to plot their performance for problems of different tightness. In the RAIN problem, tightness is the ratio of resources required for the best possible allocation (in terms of used bandwidth) divided by the total amount of resources available in the network. Since it is very hard to compute the best possible allocation, we approximate it with the best allocation found among the methods being compared.

We generated 22'000 RAIN problems, each with at least one solution. Each problem has a randomly generated network topology of 20 nodes and 38 links, and a random set of 80 demands. The problems were solved with four different strategies: *basic-SP* performs a search using the shortest path heuristic common in the networking world today, without any backtracking on decisions; *BT-SP* incorporates backtracking to the previous in order to be able to undo "bad" allocations. The next search methods make use of the information derived from the BIH: *BI-LL-HL* uses the LL heuristic for route generation and DVO-HL for dynamic demand selection, whereas *BI-LL-NL* differs from the latter in using DVO-NL for choosing the next demand to allocate. Results were computed on a Sun Sparc 60.

The probability of finding a solution to a problem in less than 1 second according to the tightness of the problems (as defined above) is given in Fig. 2. Both BI search methods perform much better than brute-force, even on these small problems, where heuristic computation (and BIH maintenance) may proportionally use up a lot of time. Noteworthy, NL outperforms HL: NL is better at deciding which demand is the most difficult to assign, and therefore achieves a greater pruning effect. The shape of the curves are similar for larger time scales. The quality of the solutions, in terms of network resource utilization, were about the same for all methods. However, when the solutions were different, bandwidth connectivity was generally better on those provided by BI methods.

Fig. 2: The probability of finding a solution within 1 second, given the tightness of the problems (22'000 random problems with 20 nodes, 38 links, 80 demands).

Note that the experimental results allow quantifying the gain obtained by using our methods. If an operator wants to ensure high customer satisfaction, demands have to be accepted with high probability. This means that the network can be loaded up to the point where the allocation mechanism finds a solution with probability close to 1. From the curves, we can see that for the shortest-path methods, this is the case up to a load of about 40%, whereas the NL heuristic allows a load of up to about 55%. Using this technique, an operator can thus reduce the capacity of the network by an expected 27% without a decrease in the quality of service provided to the customer! Moreover, according to phase transition theory, relative performance can be expected to scale in the same way to large networks.

References

- [1] C. Frei and B. Faltings. A Dynamic Hierarchy of Intelligent Agents for Network Management. In *2nd International Workshop on Intelligent Agents for Telecommunications Applications, IATA'98*, 1–16, Paris, France, July 1998.
- [2] J.W. Mann and G.D. Smith. A Comparison of Heuristics for Telecommunications Traffic Routing. In *Modern Heuristic Search Methods*, John Wiley & Sons Ltd, 235–254, 1996.
- [3] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, UK, 1993.
- [4] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE JSAC*, 14(7):1228–1234, September 1996.