SERVER ALLOCATION ALGORITHMS FOR VOIP CONFERENCING

R. Venktesha Prasad[*], *H. N. Shankar*[#], *H. S. Jamadagni*[*], *S. Vijay*[+]

[*]CEDT, Indian Institute of Science, Bangalore, India.
[#]P E S I T, Bangalore, India and NIAS, Bangalore, India
[+]Esqube Communication Solutions Pvt. Ltd, Bangalore, India.
{vprasad, hsjam}@cedt.iisc.ernet.in

ABSTRACT

Real-time services have been supported by and large on circuit-switched networks. Recent trends favour services ported on packet-switched networks. For audio conferencing, we need to consider many issues such as scalability, quality of the conference application, floor control and load on the servers. In this paper, we deal with the allocation of Conference Servers (CS) [18] that is part of an audio service framework designed to provide a Virtual Conferencing Environment (VCE). The system is designed to accommodate a large number of end users speaking at the same time and spread across the Internet. The framework is based on conference servers which facilitate audio mixing and distribution. Concurrently, we exploit the Session Initiation Protocol (SIP) capabilities for signaling purposes. We address here the problem of facilitating seamless conference amongst participants using CSs. This demands a proper allocation of CSs to clients to maximise the number of participants served and at a reduced cost. This problem is popularly identified as the Facility Location Problem, a class rich in challenges used to model the minimisation problem of assignment of clients to servers. Seeking a more realistic approach, we avoid over-simplifying assumptions; thus the problem becomes relatively harder. Since these problems are NP-hard, algorithms leading to approximate solutions or those involving heuristics are commonly resorted to. We present heuristic algorithms to solve this class of problems and bring about the effectiveness of their performance.

Keywords: VoIP Conference, Conference Servers, SIP, Facility Location Problem, Heuristic Algorithms
Areas: Converged communications (VoIP) service management, Distributed multimedia service management

1. INTRODUCTION

Todays Internet uses the IP protocol suite that was primarily designed to transport data and provide best effort data delivery. Traditional data differ from voice and video in aspects of delay-constraints and packet loss tolerance. Hence as time-sensitive voice and video applications are deployed on the Internet, the inadequacy of the Internet is exposed progressively. The aim is to port telephone services on the Internet. Virtual conference (teleconference) facility is at the cutting edge of those services. Audio and video conferencing on Internet are popular [6] for the several advantages they offer [2]. Clearly, most collaborative works demand audio conferencing more frequently than video interactions [7]. It makes sense to deal with audio conferencing first; related and more involved issues need to be tackled while supporting video.

[*]Author for correspondence

The bandwidth required for teleconference over the Internet increases rapidly with the number of participants; reducing bandwidth without compromising audio quality is a challenge in Internet Telephony. Audio "quality" in connection with a conference includes facilitating interactivity, i.e., allowing impromptu speech, and spatialism [8, 9]. The critical implementation issues are:

1. Packet delay;
2. Echo;
3. Customized mixing of audio from selected clients [8];
4. Automatic selection of clients to participate in the conference;
5. Playout of mixed audio for every client;
6. Handling clients not capable of mixing audio streams (such clients are known as "dumb clients"); and
7. Deciding the number of simultaneously active clients in the conference without compromising voice quality.

In case of large number of participants, unlike a 3-party conference, no client can be expected to handle streams from all clients. Then intermediate servers must serve to control, maintain and support a conference. Many of the issues listed above are closely related to server location *vis-à-vis* clients and assignment of clients to servers. Next we briefly explain the backdrop against which the problem tackled in this paper is set.

1.1. Virtual Conferencing Environment

We devote this section to the description of the system architecture [19, 20] in which the allocation problem is set. For details of the architecture for VoIP conference we refer to our work in [18] and furthered in [19, 20].

We do not restrict our conferencing system to work on small conferences only, but rather on large audio Virtual Conference Environments (VCE) that have hundreds (or even touching thousands) of users across a Wide Area Network (WAN) such as the Internet. This view stems from an appraisal that VCEs will gain in importance sooner than later, even as interactive audio conferences are becoming more popular thanks to the spread of media culture around the world. Two issues must be taken care of when building a VoIP conferencing system:

1. the front-end consisting of the application program running on the end-users computers; and
2. the back-end that provides other application programs that facilitate conferencing and conference service.

Participating users are grouped into several domains. These domains are Local Area Networks (LANs) such as corporate or educational networks. This distributed nature asks for distributed controlling and media handling solutions, as centralized systems do not scale up for such very large conferences. More explicitly, in each domain we can identify several relevant logical components of a conferencing facility (see Fig. 1). An arbitrary number of end users (clients) can take part in a conference at a time. Every user is included in one and only one domain at a given instant, but can move from domain to domain (nomadism). In our conferencing environment, these clients are regular SIP User Agents (SIP UAs), as defined in the RFC 3261 [15]. Use of SIP helps interoperability since SIP is the currently popular standard. These clients are thus not aware of the complex setting that supports the conference and this is highlighted below.

The back-end consists of two servers: (i) one SIP Server (SIPS) per domain and (ii) one Conference Server (CS) per domain. SIPS handles all signaling aspects of the conference (clients joining, leaving, etc. [19]) while CS is for handling media packets for the clients of the domain. Although Fig. 1 shows SIPS in each domain, there can be only one SIPS in the entire VCE. However, one CS must be present in each domain for handling clients. A CS selects four streams out of all the clients in its domain and shares them with other CSs on multicast or unicast (if multicast is not supported by the network). In every packet interval (not more than 40ms to support interactivity) each CS selects four streams out of the several received from other CSs and its own clients. These will be sent to clients in its domain, often on multicast [9, 17, 20]. Usually, a SIPS and a CS run on the same computer.

The domains and entities in a VCE as shown in Fig. 1 are not fixed apriori. Specifically, this paper deals with the problem of identifying the clients and CSs of domains (or forming such domains) for supporting a large conference. Once a client is assigned to a domain the control messages for conference setup and maintenance are between SIPS and in turn CSs. This is achieved using 'Subscribe' and 'Notification' messages provided by SIP. This is included in outband signalling (see Fig. 2). In this paper we do not delve into control messages and conference maintenance issues by SIPS servers. We focus our attention on finding a configuration of domains around possible locations of CSs; in this aspect, we seek a suitable 'near optimality'. The optimality is in terms of some of the network parameters – delay, connection cost, etc. – explained in Section 2.1.

1.2. Constraints

In the context of the foregoing on the architectural setup of the conferencing environment we must identify a working model of conferencing before we formally define the problem. There are some constraints in handling a conference in the above setting. They are listed below.

1. CSs locations are fixed.
2. CSs (these are software entities) are enabled only at such locations and only when necessary.
3. Clients are assigned to a CS as and when conference service is requested.
4. A client is assigned to only one CS.
5. Only one CS is opened at a location and if there is a provision for another CS at a particular location, it is assumed to be another location with similar parameters with respect to the network.

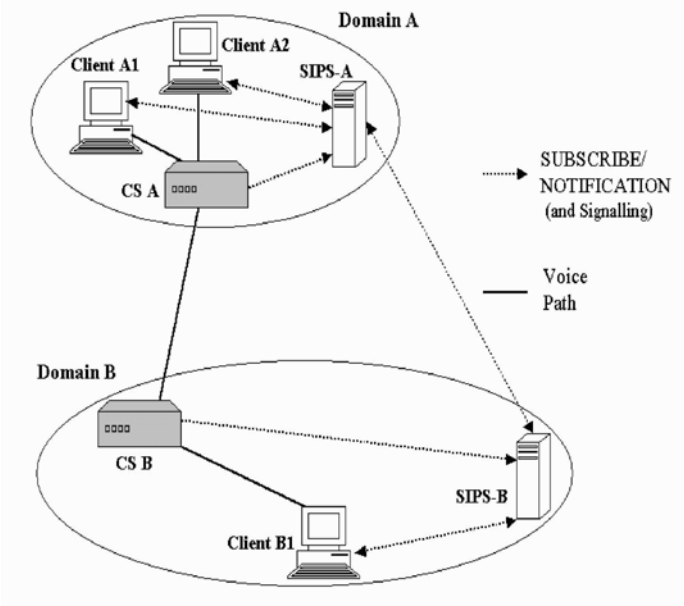

Fig. 2. Message flow between different entities (control and voice paths

The rest of the paper is organized as follows: Section 2 has a detailed discussion on concerns in providing this service along with the problem definition. It has, in addition, a brief note on the possible 'cost' formulation. Section 3 has the proposed heuristic algorithms. A discussion on the results comprise Section 4. We conclude in Section 6.

2. THE ALLOCATION PROBLEM

With respect to the above architecture, the frequently asked questions are the following.

- If a client is in a remote domain, say, a dial up link, how can that client be served?
- Should a CS be always opened in a SIPS domain or can a client be assigned to a CS in another domain?
- With several clients in a small network neighbourhood, is it cost effective to assign a new CS to that group?

These questions are addressed in the sequel. This distributed conference system based on CS is highly scalable [9, 18, 19, 20]. But it introduces the problem of allocating CSs to clients. We consider choosing of CSs and CS group formation. Scalability depends on the number of CS domains. A few CS domains cannot accommodate too many clients. With more CS domains, communication cost between CSs goes as $O(\nu^2)$, where ν is the number of domains. While finding an optimum number of CSs, the costs of client connecting to a CS and of CS opening and maintaining are to be considered.

A conference is controlled by SIPSs which are distributed [20], or by a centralized controller [9]. The servers can allocate CSs to

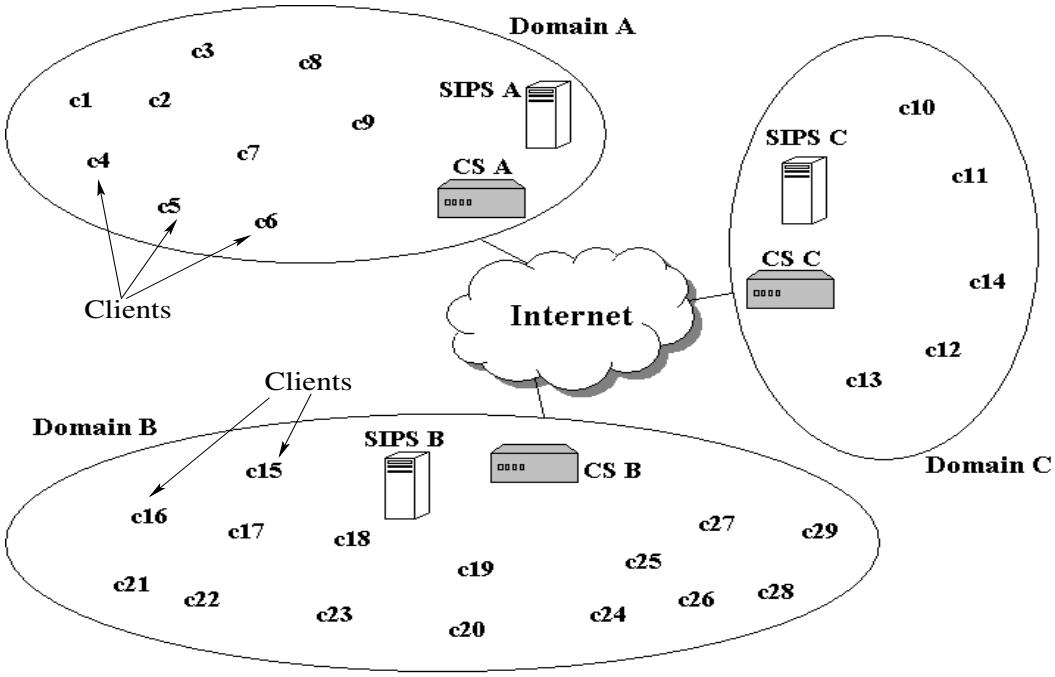

Fig. 1. ***VCE*** **architecture showing domains, clients and servers**

clients before the start of a conference if the clients in that conference are known apriori. That is, the conference is booked by a party with a list of participants to be invited at a specific hour. In this case allocation is done in the beginning of the conference. If clients are successively invited to an ongoing conference, then SIPSs have to reallocate CSs to clients. There are two ways to go about this:
(1) Only the new client is allocated to that CS which reduces the cost;
(2) The new client is allocated to a nearest CS and periodically, all clients are allocated afresh to CSs available, thus reducing the overall cost of computation.

We propose to take the second approach to avoid too frequent changes in allocation. In VoIP conferencing, allocation of a CS to a client implies that the client should send and receive packets to and from that CS. This involves changing the CS address stored at the clients. This is simple to handle because the UDP packets that carry voice data are to be sent to a different CS address. Since UDP is not connection oriented we may effect the change by suitably changing the destination address in the database.

Allocation of clients to CSs in the present setting is popularly known as the Facility Locator problem. In Operations Research, facility location problems deal with finding the optimum number of shops to be opened so that the total cost including opening cost of shops and transportation cost for customers is minimised. Here this is equivalent to CS maintaining cost and clients connecting costs. This class of problems is known to be NP-hard [1, 4, 5].

There are three established ways of solving these problems [3].

1. Construct algorithms taking exponential time for the most difficult instances but find the optimum fast enough for typical instances, e.g., Branch and Bound and Dynamic Programming.
2. Heuristics provide excellent solutions very quickly if they are admissible; however, they do not guarantee a solution.
3. Approximation algorithms relaxing the optimality requirement provide a solution within some factor of polynomial time guaranteeing a reasonable running time and a good solution on even the most difficult instances.

In practice heuristics run faster than the other two. The problem in this paper (Section 2.2) does not have any known approximation algorithm.

2.1. Cost

Before formulating the CS allocation (or location) problem, some discussion on the cost of service will be useful. Some knowledge of the entire network and locations of clients and CSs is necessary for an overall acceptable assignment of clients to CSs. Hence it is assumed here that some information of the parameters of the link connecting clients to CSs is available. Depending on the specific needs, the cost of connection between a client and a CS can be assigned in terms of number of hops (a measure of network remoteness), link bandwidth and cost of the physical connection. In case the link is a leased line connecting to Internet, the cost of using that link can also influence computation of the overall cost. Cost is not specifically defined in the following discussion but some specific cost based on the above parameters is adopted. Similarly, hardware and location form the basis for computing the cost of opening a CS. These two costs – cost of connection between a client and a server, and cost of opening a CS – are used to find the allocation of clients to CSs.Though the costs taken here are conceptual, the cost can also be estimated by using explicit network measurement techniques as in [21]. Since, in this paper, we address primarily the effectiveness

of our heuristic algorithms, we reserve the discussion and applicability of these measurements for cost calculation for other venue.

A CS has to incur some computation for each client assigned to it depending on whether the client is a soft client or a hard client, the coding used by the client and whether mixing is required at the clients. These overheads, called "demands" from clients, can be taken as units of computation. Moreover, CSs have limited computation capacity. Further, a CS with a dedicated computation engine such as DSP processor serves more clients than the one running on a general purpose computer. To discourage opening of a CS at a location, cost of opening that server can be made high. Similarly, the cost of connecting a CS to a remote client can be made large. These are the engineering choices that can be made while constructing a problem instance. We have considered the most general case in this paper to find the effectiveness of our algorithm.

2.2. Problem Formulation

Let $F_S = \{1, 2, \ldots, m\}$ be the set of m CSs situated at preidentified locations. Let $D_C = \{1, 2, \ldots, n\}$ be the set of n clients whose locations are known. Let the cost of connecting a client $j \in D_C$ to a CS $i \in F_S$ be c_{ij}. The notion of cost follows the discussions in Section 2.1. Let d_j, $j \in D_C$, be the demand (in the form of computation requirement, in integer units) from Client j and let $f_i,\ i \in F_S$ be the cost of opening CS i. Let l_i, $i \in F_S$, be the limit of CS i in number of units of computation that it can afford.

Now the client allocation to CSs can be expressed as an integer program, with x_{ij} and y_i as $0-1$ integer variables. $y_i = 1, i \in F_S$, indicates that the CS i is opened and $x_{ij} = 1,\ i \in F_S,\ j \in D_C$, implies that the Client j is assigned to CS i. The minimisation problem (P_{min}) is formulated as an integer program as

$$\min: \quad \sum_{i=1}^{m} f_i y_i + \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

subject to

$$\sum_{j=1}^{n} d_j x_{ij} \leq l_i, \ \text{for each } i \in F_S \tag{2}$$

$$x_{ij} \leq y_i, \ \text{for each } i \text{ and } j,\ i \in F_S;\ j \in D_C \tag{3}$$

$$\sum_{i=1}^{m} x_{ij} = 1, \ \text{for each } j \in D_C \tag{4}$$

$$x_{ij} \in \{0, 1\} \ \text{for each } i \in F_S,\ j \in D_C \tag{5}$$

$$y_i \in \{0, 1\} \ \text{for each } i \in F_S \tag{6}$$

Constraint 2 limits the number of clients connected to a CS depending on its limit. Constraint 3 ensures that if there is an assignment of a client to a CS then that CS must have been up and running. Constraint 4 ensures each Client j is assigned to a CS $i, i \in F_S$ and $j \in D_C$. Constraints 5 and 6 are part of integer programming. Another constraint that a client is allocated to only one CS is captured in the above constraints and hence not mentioned explicitly. This constraint is termed 'unsplittable' demands in operations research terminology. Note that the second term in (P_{min}) is not multiplied by d_j, as is usually done if facilities are shops and c_{ij} are customer locations to cover cost of transportation.

3. SOLUTION BY HEURISTICS

This is a hard problem since there is no polynomial solution known [10]. There are actually two problems: (a) assigning clients to CSs with less cost; and (b) opening least number of CSs so that there is saving in supporting the conference service. However, the assignment and number of CSs to be opened are to be found so that overall cost is minimised. In its simplest form (where demands may be split and with or without a limit on capacity of CSs (servers)) this problem may be solved using approximation algorithms. Some solutions consider unit demands. When demands are splittable, i.e., they can be met by more than one server, solutions using transhipment algorithms to assign the clients are suggested. These solutions and their variants are covered in [11, 12, 13, 14]. When the CSs have limits and clients are to be served by one CS (unsplittable demands), it is NP-hard to compute the minimum assignment even if the optimum number of CSs and their locations are provided as input [14].

In this problem of allocating CSs, there are instances in which a solution may not exist. A case in point is when the total capacity of CSs is less than the total demand by clients. In these cases heuristic algorithms definitely fail. Heuristic algorithms try to find a near optimal solution, if there exists one. The difference between the solution obtained using a heuristic algorithm and an optimal solution (found using, say, Integer Programming (IP) formulation of the problem) is unbounded.

3.1. Algorithms: Phase-I

The algorithm suggested here is divided into two phases. The first one is the assignment phase. Here the clients are assigned to CSs in set $F_S^G \subseteq F_S$. CSs only in F_S^G are considered; the algorithm considers CSs outside this set as nonexistent. In the second phase, the number of CSs is minimised by closing some of them and clients previously assigned to the recently closed CSs are reallocated to other CSs. Algorithms in both phases are greedy. Algorithms of the first phase are given below[1].

Algorithm 3.1 ***Assignment of clients to given set of CSs (F_S^G)***

Arrange c_{ij} in the list $L(k)$ in ascending order, where $k = \{1, 2, \ldots, mn\}$. Each k corresponds to a pair (CS i, Client j).
Consider assignment of clients to a CS as $A_C[\cdot]$, a row vector that holds the CS identity i for each Client j.
$A_C[j]$ is zero vector to start with.
$k = 1$. REPEAT
{

Step 1: *Assign the Client j to CS i that corresponds to k of $L(k)$ if*

- *For CS i, $l_i \geq d_j$, and*
- *Client j is not already assigned;*

Step 2: *$l_i = l_i - d_j$, if Client j is assigned to CS i. Mark Client j as assigned to CS i by updating $A_C[j] = i$ and $x_{ij} = 1$.*

Step 3: *Stop if all clients are assigned; Return the total assignment cost (that is, $\sum_{i,j} c_{ij} x_{ij}$ and $A_C[j]$. Break;*

[1] We adopt a descriptive style for presenting the algorithms with a view to render them more reader-friendly.

Step 4: *Return '$Infeasible$' if all clients are not assigned when $k = mn$.*

Step 5: $k = k + 1$

}

This Algorithm 3.1 acts on the given set of CSs, F_S^G, and uses $[c_{ij}]$ matrix to assign clients to CSs. Algorithm 3.1 provides the popular greedy solution wherein the client nearest to a CS is assigned first. This heuristic may give an optimal solution in some instances, as in if there exists a Monge sequence [16].

The primary computation in this algorithm is sorting all the elements of the cost matrix $[c_{ij}]$. That is mn elements. The worst case complexity of this operation is m^2n^2. The next step is to check whether a client is assigned to a CS out of D_C clients. The worst case complexity of this algorithm is easily found to be $O(m^2n^2)$. The advantage of this algorithm is that it is very easy to implement. The next algorithm is an intelligent assignment solution.

Algorithm 3.2 ***Assignment of clients to given set of CSs (F_S^G)***

`Consider matrix` $[c_{ij}]$ `wherein each row, i, represents a CS and each column j represents a client.`
`Consider assignment of clients to a CS as` $A_C[\cdot]$`, a row vector that holds the CS identity i for each Client j.`
$A_C[j]$ `is zero vector to start with.`

Step 1: *Find $b(j) = \min_i \{c_{ij}\}$ for each $j \in D_C$. That is, finding the CS $i \in F_S^G$ that results in minimum cost of assignment to Client j. $b(j)$'s have the minimum cost of assigning Client j to a CS.*

Step 2: *Update $A_C[j] = i$ by assigning Client j to CS i that corresponds to the minimum cost found as above and set $x_{ij} = 1$.*
Note: If two positions in $[c_{ij}]$ have minimum value, the first one is chosen.

Step 3: *Find set of CSs, $F_S^{(OL)}$, that are overloaded taking into account $A_C[j]$ and l_i. If there are no overloaded CSs, goto Step 7.*

Step 4: *Find difference matrix $[a_{ij}]$ where $a_{ij} = c_{ij} - b(j)$, $i \in F_S^G$, and $j \in D_C$. Mark 'X' in $[a_{ij}]$ if Client j is allocated to CS i in Step 2.*

Step 5: *For each CS $r \in F_S^{(OL)}$,*
for clients $s \in D_C^{(r)}$, i.e., clients assigned to the CS r, find new CS-client pair (p, q) that has the co-ordinates of $\text{argmin}_{(i,s)}\{a_{is}\}$,
(i.e., position of $\min\{a_{is}\}$ over i and s), where $s \in D_C^{(r)}$, $i \in F_S^G$ and $i \notin F_S^{(OL)}$ and CS p is able to serve this new client without overloading itself (a_{ij}'s marked with 'X' are not to be considered)

Step 6: *If a new CS is not found in Step 5,*
Return 'Infeasible'.
Else, assign Client q to CS p, i.e., $A_C(q) = p$ and $x_{pq} = 1$.
Subtract the load d_q from CS r.
Update $A_C[j]$ and $F_S^{(OL)}$ taking into account this change. Mark a_{rq}, the previous position of Client q, with 'X'.
Repeat Step 5 till $F_S^{(OL)} = \{\}$.

Step 7: *Return $A_C[j]$ and cost of assignment $\sum_{i,j} c_{ij}x_{ij}$.*

Heuristic Algorithm 3.2 does an intelligent selection of the CSs unlike Algorithm 3.1. Minimum in each column of matrix $[c_{ij}]$ is found first. This minimum of a column is subtracted from other elements of the same column of $[c_{ij}]$. This forms the difference matrix $[a_{ij}]$. In Step 2, clients are allocated to CSs considering only the minimum cost of connecting Client-j to Server-i from $[c_{ij}]$ (i.e., the (i, j) pertaining to 0 of each column in $[a_{ij}]$) without checking whether a CS is overloaded. If no server is overloaded, the assignment is successful and also optimum. If not, some clients from the overloaded CSs are to be moved to CSs that are not loaded to their limits. This is done with the aid of $[a_{ij}]$ matrix. Without loss of generality, let us assume that Server-r is overloaded. Then we find all the clients assigned to that server. Let these clients forms a set C_r. Then for all the corresponding columns of clients in C_r, find the minimum entry in $[a_{ij}]$. Let this correspond to the (p, q)th entry. Now, move the Client-q to the Server-p. However, before moving now, we take care that moving the client does not overload the Server-p. Reassignment is done with small, hopefully least, increase in cost.

The primary goal of this algorithm is in finding the minimum element of each column of the cost matrix $[c_{ij}]$. That is out of m elements. The worst case complexity of this operation is mn. The next step is to find the difference matrix $[a_{ij}]$, the complexity of which is also of the order mn. Now suppose that all the clients are assigned to one server that cannot serve even one client (*i.e.,* the minimum demand over all clients exceeds the capacity of the CS). Then we need to iteratively search for the minimum of $(m - 1)n$ elements. This will continue till finding a minimum n times over all n clients out of $(m - 1)n$ entries. Therefore the worst case complexity of this algorithm is evidently $O(mn^2)$. We note that the complexity has reduced by an order of magnitude. However, it is nontrivial to code and implement this algorithm.

> **Remark:** We cannot assert as to which one of the above algorithms performs better than the other *always*. Such an assertion is clearly problem-dependent. It is with this in view that we have proposed here alternative heuristic approaches. The general strategy is therefore to invoke the two algorithms, one after the other, and take an allocation which is the minimum of the two at each iteration over to the next phase.

3.2. Algorithm: Phase-II

This is the final algorithm that tries to minimise the cost of opening the CSs. First we find an allocation of clients to the servers, and later we try to close CSs one by one using the cost function

$$T_i = \frac{f_i}{l_i} - \sum_i \sum_j I_C(j) c_{ij} \tag{7}$$

This cost function takes into account the cost of operating the server *vis-à-vis* its capacity (first term in Eq. 7). Later we close the Server-L corresponding to highest T_L.

Algorithms 3.1 and 3.2 in the allocation (assignment) phase can be used individually or each one may be applied on an instance and the best possible solution used. As noted above, depending on the problem instance one algorithm may perform better than the other in the Phase-I. These two algorithms are used to solve the problem P_{min} in this phase (Algorithm 3.3), which is our main heuristic algorithm.

Algorithm 3.3 ***A heuristic solution to the problem (P_{min})***

Let $F_S^{(O)}(i)$ be a vector that represents whether a CS is open. It is found using client assignment vector $A_C[j]$.

$$F_S^{(O)}(i) = \begin{cases} 1 & \text{if CS } i \text{ is serving any client} \\ 0 & \text{Otherwise} \end{cases}$$

Total cost $SOLN$ of any assignment is found using

$$SOLN = \sum_i F_S^{(O)}(i) f_i + \sum_{\forall i,j} I_C(j) c_{ij} \tag{8}$$

$$\text{where} \quad I_C(j) = \begin{cases} 1 & \text{if Client } j \text{ is served by CS } i \\ 0 & \text{Otherwise} \end{cases}$$

Note: The second sum is nothing but $\sum_{i,j} x_{ij} c_{ij}$

Step 1: *Initialise $A_C[j]$ and $F_S^{(O)}(i) \ \ \forall i, j$ to zero, $F_S^G = F_S$ and a set $Z = \{\}$.*

Step 2: *Find $A_C[j]$, $F_S^{(O)}(i)$ corresponding to a minimum $SOLN$, after using the two Algorithms 3.1 and Algorithm 3.2 of first phase.*

Step 3: *Find T_i, $i \in F_S$, as defined below and arrange in descending order.*

$$T_i = \frac{f_i}{l_i} - \sum_i \sum_j I_C(j) c_{ij}$$

Step 4: *For $k = 0$*
Do $\{$

Temporarily close CS u that corresponds to the highest T_i, and $u \notin Z$
(a) Find the best $iSOLN$ using Equation 8 out of two returned assignments $A_C[j]$'s of two first phase algorithms with $\{F_S^G - \{u\}\}$ as CSs open for assignment. (Here $iSOLN$ means intermediate solution.)
If ($iSOLN < SOLN$)
then permanently close CS u and $F_S^G = F_S^G - \{u\}$;
$SOLN = iSOLN$;
Recalculate T_i as above with CS u closed and arrange in descending order;
Else, $Z = Z \cup \{u\}$;
(b) $k = k + 1$;

$\}$Repeat till $k = m$;

Step 5: *$A_C[j]$ of $SOLN$ corresponds to the best assignment found by this algorithm. The total cost is $SOLN$.*

Algorithm 3.3 is greedy. We iteratively invoke the assignments to the CSs which are the remaining CSs after closing some of them using Eq. 7. This algorithm takes into account two first phase algorithms and the best assignment of the assignments returned by them is considered at each stage (see Step 2 and Step 4). There can be many variants of this algorithm where individual or both the first phase algorithms may be used. The complexity of this algorithm is m times the complexity of the first phase algorithm. For example if we use both the first phase algorithms, and taking best allocation returned by them, the resulting complexity is $O(m^3 n^2)$ since Algorithm 3.1 has the highest complexity. If we use only Algorithm 3.2 in Phase-I then it would be $O(m^2 n^2)$. Since the heuristic algorithm does not guarantee optimal allocation always, it may be suggested here that it is advisable to use as many algorithms as possible.

4. RESULTS AND DISCUSSIONS

Since Linear Programming (LP) formulation relaxes the constraints that x_{ij} and y_i be integers, the solution found by LP is the best solution. Integer LP (ILP) is hard and if there is a solution, it costs greater than LP solution. LP formulations are complicated [14], though not NP. It is interesting to see how Algorithm 3.3 fares for some test samples.

We generated the test cases randomly. The cost $c_{ij} \in [1, 1000]$ is generated arbitrarily. So also, $d_j \in [1, 5]$ and $f_i \in [1, 1000]$ are chosen. l_i are changed across test samples as per Table 1 (shown here a sample of many test cases).

> *Remark*: We use random inputs to test the robustness of our algorithms. The use of triangular inequality on the physical distance between CS and a client may reduce the complexity of the problem. However, we refrain from doing so as the overall cost so characterized would then be necessarily blind to installation costs and service costs in a packet switched network. For example, two clients with different ISPs (at the same geographic location) may be billing differently. In general, we can see higher density of clients in geographic proximities (and hence network distance) as, for example, conference participants in local pockets of business centers. In these cases our algorithm converges faster than the cases with random inputs.

The optimal cost of the solution (ILP formulation) for the problem (P_{min}) lies between that of LP and of Algorithm 3.3. We have solved the LP formulation using the freely available 'lp_solve' program for Linux OS. Table 1 shows the ratio of performance of LP to that of Algorithm 3.3. So the difference between the solutions of Algorithm 3.3 and ILP would be even less than what is shown in Table 1. Table 1 is not an average value of many test samples but tries to give an overall picture for different test scenarios.

Algorithm 3.3 is implemented in two different forms differing in the method of assignments in the first phase. They are: (a) Algorithm 3.1 alone; and (b) The better of Algorithm 3.1 and 3.2. We have not shown Algorithm 3.2 alone in Table 1 since when we use Algorithm 3.1 or Algorithm 3.2 alone the result varies with the test cases. This is testimony to our earlier remarks that "it cannot be asserted that one of the algorithms in Phase-I is preferred over

Table 1. Heuristic Algorithm 3.3 compared with LP solution

Number of Selectors	Number of Clients	Range of l_i	LP Solution	Heuristic Algorithm 3.3 with Phase-I Algorithm(s)		Ratio of Heuristic to LP
				Algo 3.1	best of (Algo 3.1 & Algo 3.2)	
10	50	1-50	9707	10309	10309	1.062
10	100	1-50	13000.4	13207	13037	1.003
10	150	1-150	20439.6	22620	21155	1.028
10	500	1-500	53541.5	57047	53876	1.005
10	1000	1-1000	99698	102049	99951	1.003
10	1000	1-800	100376	112422	101813	1.014
10	1000	1-1000	105684	127217	111214	1.052
10	1200	1-1000	174134	179141	174443	1.002
12	1000	1-1000	93761.8	102985	94732	1.011

the other in all cases. Consequently it is profitable to use as many heuristics as possible".

The lower bound of the solution provided by the approximation algorithms for the uncapacitated facility location problem is shown to be 1.463 times that of the optimal solution [5]. An algorithm given in [12] shows an average case solution of 1.03 and worst case solution of 1.05-1.07 as corresponding values for various test cases, again for the uncapacitated case. Algorithms given here are for capacitated constrained problems. Table 1 shows that even for problems of large scale the error is comparable to the results of the uncapacitated version. It can also be seen that with two heuristics used for assignment, Algorithm 3.3 achieves results very close to the optimum solution compared to using Algorithm 3.1 alone for assignment (first) phase. The heuristic may not give a solution, though one exists. Yet heuristic algorithms serve as a good starting point to group clients. Moreover, with many heuristics the possibility of finding a solution, when one exists, is enhanced.

Now we shall get back to the question of how these algorithms are used in our allocation of CSs. We suggest that these algorithms be implemented on SIPS in each domain. Once they receive information about a new client added to the conference, the algorithm is invoked. This requires information regarding the new column in $[c_{ij}]$ and the demand from the new client. Since, this information can be exchanged across SIPS while supporting the conference [19] unique allocation can be found at every SIPS. If the conference is booked earlier this algorithm can be used only in the beginning.

The allocation found using the above heuristics should answer the questions raised in Section 2. Likelihood of success of allocation can also be increased by some pragmatic assumptions. For example, in case of an isolated client, the nearest CS can serve it and multiple streams from CS to client can be mixed at that CS to reduce the bandwidth. There are some open issues here. One such issue concerns the method of implementing this algorithm in the proposed distributed conference setup [20, 19].

5. SOME RELATED SOLUTIONS

There are many proposals [22, 23, 24] to handle multiparty interactions in the absence of availability of *multicasting* in Interenet with reliability. They usually solve this by hierarchically configuring the individual nodes as relay servers (Rendezvous Peers) in the network for distributed multicast support at the application layer. To mention a few, End System multicast (ESM) and Overlay Peer to Peer networks [24], (for example, jXTA [25] and Pastry [26]) try to overcome the non-availablity of multicast support in Internet. In a complete P2P ESM or PeerCast ESM networks [28] there can be many Rendezvous Peers (which are some what similar to CSs of our Architecture in functionality) between source and end nodes resulting in higher latency. In our architecture, we limit only two CSs in the voice path to have a low latency which is a necessity for interactive conversations. Therefore, we need form clusters of clients around CSs to to reduce the latency un like hierarchical Rendezvous Peers.

Since the application here is *interactive* voice conferencing, we allow upto four participants speaking at the same time [9] out of many participants. Moreover, the set of active speakers can change dynamically unlike Streaming/Broadcast over ESM or Overlay networks where predominently only one source generates streams. Thus we need to properly identify the clusters of clients and their allocation to CSs for this application. In case of ESM networks there are ways to balance the load on Rendezvous peers and minimize the cost of connectivity [27, 28]. We feel that our algorithms can be tuned and applied to work even in the ESM or Overlay network environments since we have formulated and solved a generalized allocation problem.

6. CONCLUSIONS

The problem considered here is a hard problem that has not been attempted in literature without relaxing some of the constraints. The heuristic algorithms presented here attempt to solve hard problems involving (i) assignment of clients to CSs that do not have unit demands and hence not solvable using transportation problem; and (ii) reducing the total cost by opening an 'optimum' number of CSs. The algorithms proposed here have been shown to perform 'close enough' to LP solution, thus they are nearer to the optimal solution (i.e., IP formulation). They can be deployed easily. They are portable to the general domain of Facility Location problems though we have discussed here a specific case of VoIP application.

Our Algorithm is generic enough so that it can be used in more general cases of facility locator problems as well as a host of ESM or Overlay Network applications such as this. We have not used measured numbers regarding the cost matrix at present. Work on these aspects is in progress. Future directions also include a thorough study of a few pilot runs on the Internet. And, tuning of these algorithms for online applications when clients join and leave at will as in the case of participants in a real conference hall in quick successions.Many variants of heuristic algorithms of both the

phases have been worked out; these are outside the scope of this paper.Future directions include a thorough study of a few pilot runs on the Internet. Also, tuning of these algorithms for online applications when clients join and leave at will as in the case of participants in a real conference hall.

7. REFERENCES

[1] "A compendium of NP optimization problems," in (P. Crescenzi and V. Kann, eds.)", http://www.nada.kth.se/ viggo/problemlist/compendium.html.

[2] Amitava Dutta-Roy, "Virtual Meetings with desktop conferencing", *IEEE Spectrum*, July 1998, pp. 47-56.

[3] N. Edwards, "Improved approximation algorithms for the k-level facility location problem" *PhD thesis, Cornell University*, Ithaca, NY, 2001.

[4] M. R. Garey and D. S. Johnson, "Computers and intractability." Freeman, San Francisco, 1979.

[5] S. Guha and S. Khuller, "Greedy strikes back: Improved facility location algorithms," *Journal of Algorithms*, vol. 31, pp. 228–248, 1999.

[6] Lisa R. Silverman, "Coming of Age: Conferencing Solutions Cut Corporate Costs", *White Paper*, www.imcca.org/wpcomingofage.asp

[7] E. Doerry, "An Empirical Comparison of Copresent and Technologically-mediated In-teraction based on Communicative Breakdown", *PhD thesis*, Graduate School of the University of Oregon, 1995.

[8] M. Radenkovic and C. Greenhalgh, "Multi-party distributed audio service with TCP fairness", in Proceedings of the 10^{th} *ACM International Conference on Multimedia (MM 02)*, Juan-les-Pins, France, pp. 1120, December 2002.

[9] R. Venkatesha Prasad, "A New Paradigm for Audio Conferencing on Voice over IP (VoIP)", *Ph.D Thesis, Indian Institute of Science*, Bangalore, India, 2003.

[10] C. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity* Prentice Hall, India, 2001.

[11] Martin Pal and Eva Tardos and Tom Wexler, "Facility location with hard capacities," in *Proceedings of the 42nd Annual IEEE Symposium on the Foundations of Computer Science*, 2001.

[12] M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani, "A greedy facility location algorithm analyzed using dual-fitting," in *In Proceedings of 5th International Workshop on Randomization and Approximation Techniques in Computer Science*, vol. 2129, pp. 127–137, 2001. Lecture Notes in Computer Science.

[13] K. Jain, M. Mahdian, E. Markakis, A. Saberi, and V. Vazirani, "Greedy facility location algorithms analyzed using dual fitting with factor-revealing LP," *Journal of ACM*, 2002.

[14] M. R. Korupolu, C. G. Plaxton, and R. Rajaraman, "Analysis of a local search heuristic for facility location problems," *ACM symposium on Discrete Algorithms*, pp. 1–10, 1998.

[15] J. Rosenberg, H. Schulzrinne et al., "SIP: Session Initiation Protocol", *RFC 3261*, IETF, June 2002.

[16] R. Shamir and B. Dietrich, "Characterization and algorithms for greedily solvable transportation problems," *ACM symposium on Discrete Algorithms*, 1990.

[17] R. Venkatesha Prasad, Joy Kuri, H. S. Jamadagni and Ravi Ravindranath, "A Scalable Architecture for VoIP Conferencing", in *Journal on Systemics, Cybernetics and Informatics (SCI)* , Vol. 1, No. 11, 2003.

[18] R. Venkatesha Prasad, Richard Hurni, H S Jamadagni, "A Scalable Distributed VoIP Conferencing using SIP", *Proc. of the eigth IEEE Symposium on Computers and Communications*, Antalya, Turkey, June 2003.

[19] R. Venkatesha Prasad, Richard Hurni, H S Jamadagni, "A Proposal for Distributed Conferencing on SIP using Conference Servers", *Proc. of MMNS 2003*, Belfast, UK, September 2003.

[20] R. Venkatesha Prasad, Richard Hurni, H. S. Jamadagni, H. N. Shankar, "Deployment Issues of a VoIP Conferencing System in a Virtual Conferencing Environment", *ACM symposium on Virtual Reality and Software Techniques*, Osaka, Japan, October 2003.

[21] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. "Construction of an efficient overlay multicast infrastructure for real-time applications". *Proceedings of INFOCOM*, 2003.

[22] M. Castro, M. Jones, A. Kermarrec, A. Rowstron, M. Theimer, H. Wang and A. Wolman, "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays" *IEEE INFOCOM* 2003.

[23] Y. H. Chu, S. G. Rao and H. Zhang. A Case for End System Multicast, *Proc. of ACM SIGMETRICS*, June 2000.

[24] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," *18th ACM SOSP*, Oct. 2001.

[25] Project JXTA *http://wwws.sun.com/software/jxta/*

[26] A. Rowstron and P. Druschel, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems". *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Heidelberg, Germany, pp. 329-350, November, 2001.

[27] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr. "Overcast: Reliable multicasting with an overlay network", *Proceedings of OSDI*, 2000.

[28] Jianjun Zhang, Ling Liu, Calton Pu, Mostafa Ammar, "Reliable End System Multicasting with a Heterogeneous Overlay Network", GIT-CERCS-04-19, Center for Experimental Research in Computer Systems (CERCS), Georgia Tech, 2004.