Server Selection with Delay Constraints for Online Games

Yuh-Rong Chen, Sridhar Radhakrishnan, Sudarshan K. Dhall
School of Computer Science
University of Oklahoma
{yrchern, sridhar, sdhall}@ou.edu

Suleyman Karabuk
School of Industrial Engineering
University of Oklahoma
karabuk@ou.edu

***Abstract*—Improving latency is the key to a successful online game-playing experience. With the use of multiple servers along with a well-provisioned network it is possible to reduce the latency. Given a network of servers, game clients, and a desired delay bound, we have designed algorithms to determine the subnetwork of servers whose cardinality is minimal. We have considered the cases wherein the subnetwork architecture is a client-server and a peer-to-peer. We have also provided exhaustive empirical evaluations of our algorithms and compared their performance with the optimum. Experimental results show that our polynomial-time algorithms could find good solutions quickly.**

I. Introduction

Networked Virtual Environments (NVEs) such as Massively Multiplayer Online Role Playing Games (MMORPG) wherein a number of users interact with each other through the Internet have become commonplaces. While the number of users is growing, technological challenges arise with real-time constraint being one of them. Realtimeness is one of the important factors related to game-playing experience. For example, when a player performs an action in a game, the action must take effect within a short period of time, otherwise users may stop playing and leave the game [1], [2].

Researches have been done on traffic analysis and modeling [19] and user behaviors under different network quality [2]. There are also researches on proposed architectures for huge virtual environments or online games [11], [9], [15]. Event synchronization protocols which are important to maintain a consistent game are proposed in [4] [6].

In this paper, we present several heuristic server selection algorithms (subnetwork construction) utilizing the well-provisioned server networks that take into consideration Client-Server architecture (Algorithm 1, 2), Peer-to-Peer architecture (Algorithm 3) and the delay constraints. This type of problems are a type of *steiner tree problems* with additional constraints and their extension. Our algorithm for finding a peer-to-peer subnetwork with a desired delay bound guarantees to find such networks for a large classes of these bounds. We show that the that *Zoom-in Zoom-out* (ZIZO) technique by Lee et. al [10] can sometimes fail to find such subnetworks for delay bound for some of these classes of bounds.

The paper is organized as follows. In Section II, we discuss the system model and problem formulations. Section III describes our proposed server selection algorithms. The performance of proposed algorithms is given in Section IV. Conclusions are drawn in Section V.

II. System Model and Problem Formulations

A. Types of Communication

A *node* in a network is a computing system that participates in communication and computational activities relating to the game. In the following discussion, we refer to a *client* as a computer with the software installed for playing the game that renders and presents the game states to the user [10].

The network architecture for the online game dictates the mechanisms to maintain the game states. Generally it could be classified into two types according to the communication models: *client-server* (centralized) and *peer-to-peer*.

Consider the scenario in which all clients must obtain the state of the game. In a client-server architecture, all events generated by the clients are sent to a central server first. Then the central server computes the new state of the game and sends necessary updates to the clients. In this architecture, we define the latency to update an event to be the maximum time difference between generation of an event and propagation of the resulting game state to all the clients. Most FPS games or MMORPGs such as Quake 4 [17] and World of Warcraft [21] use this approach. Note that in this architecture, the nodes and links involved form a tree with the central server as the root.

In a peer-to-peer approach, the users play the game *without* a central server. All the messages are exchanged between the participants directly and the new state of the game is computed at each client. Hence we define the latency as the largest communication delay between each pair of clients. Strategy games such as StarCraft [18] use this approach.

Both architectures require some synchronization mechanism to restore the order of the events. However, a peer-to-peer approach is less-scalable due to the direct message exchange and requires more sophisticated synchronization methods to maintain a persistent game state due to the lack of a central control [5].

There are other architectures. Multiserver architecture is a variation of the client-server architecture. It is usually implemented in MMORPG with each server being responsible for a portion (e.g., a region) of the game [7]. Mirrored server architecture is a hybrid architecture consisting of client-server

architecture and peer-to-peer architecture [3] [4] [6]. Multiple mirrored servers are deployed geographically instead of a single server. A client could pick one of the mirrored servers to join the game. These mirrored servers cooperate and run a synchronization protocol to maintain a consistent game state.

We mainly focus on the basic architectures (client-server and peer-to-peer) since our concern is the realtimeness. The measurement of realtimeness for other architectures are the same as the two basic architectures.

B. System Model and Terminologies

We borrow the terminologies and notations from [10], [20] to describe Server Selection Problems (SSP).

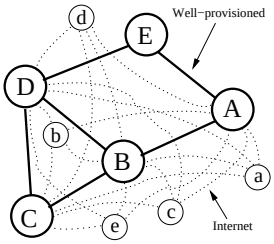

Fig. 1. An example of a game session with the set of servers $S = \{A, B, C, D, E\}$ and the set of clients $C_g = \{a, b, c, d, e\}$. The bold lines represent the well-provisioned network between the servers and the dotted lines represent Internet links between the clients and the servers.

1) Server Network: Server network has been modeled as an overlay network [13] [14] [15] on the existing Internet or as a private network dedicated to the game [10]. We use the second assumption and assume that our server network is a well-provisioned network capable of providing low latencies on its links. We use an undirected weighted graph $G = (S, E, w)$ to denote this network. The major advantage of such network is that the game provider will be able to use custom protocols and routing algorithms on the network.

Let $S = \{s_1, ..., s_n\}$ be the set of n servers. A central server refers to an entity that is capable of collectively maintaining a persistent state of an instance [10]. We also assume that a server also has the capability to route the packets to their destinations which could be another server or a client. The servers are distributed geographically and connected through a set of well-provisioned links E with the edge latency function $w : E \rightarrow \Re^+$. An example is shown in Fig. 1 with $S = \{A, B, C, D, E\}$ and $E = \{(A, B), (A, E), (B, C), (B, D), (C, D), (D, E)\}$.

In practice, each of the server could be formed by a set of computers and routers to achieve the functionality and located at different ISPs for the network requirements.

2) Clients: The clients are formed by the set of players involved in the same game session. The number of clients in a single game session is typically between 25 and 80; examples include World of Warcraft raid or battleground [21]. Large number of game sessions are in execution at the same time and the length of each game session depends on the type of the game. A typical World of Warcraft raid could take 3 to 4 hours with minor changes to the membership, on the other hand, a typical battleground lasts 30 minutes to 2 hours. We assume the length of a game session is long enough so the users could take benefit of the pre-arranged server communications. We use $C_g = \{c_1, ..., c_m\}$ to denote the set of m clients (also distributed geographically) participating in the same game session g. In the same example, $C_g = \{a, b, c, d, e\}$.

3) Accessing the Servers: We assume that there exists an *Internet* link (which has higher latency w.r.t. E) between each pair of server s_i and client c_j with the latency $d_c(c_i, s_j)$. Let $B = \{(c_i, s_j) | c_i \in C_g, s_j \in S\}$ be the set containing all the links between the clients and the servers. Note that in a game session with n servers and m clients, $|B| = m \times n$. In addition, the sets S, C and B could be visualized as a complete bipartite graph with disjoint vertex sets C of users, S of possible contact servers and B are the edges between the two disjoint vertex sets. In Fig. 1 example, $B = \{(x, Y) | x \in \{a, b, c, d, e\}, Y \in \{A, B, C, D, E\}\}$.

In the client-server model, one of the server nodes will be chosen as the central server while other servers as routers. The servers are connected by a well-provisioned network. Now all the clients can connect (a) directly to the central server through an Internet path, or (b) to some other sever (again through an Internet path) which in turn connects to the central server through a path on the well-provisioned network. Because of the presence of the well-provisioned network, it is possible that latency from (a) can be more than that of (b). An example is shown in Fig. 2. Let B be the central server, the latency for client a is 55 in case (a). It is reduced to 50 in case (b).

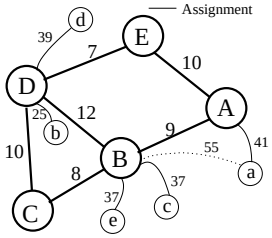

Fig. 2. An example of an assignment in Fig. 1. The numbers denote the latencies of links.

4) Assignments: In order to participate in a game session, each client c_i is connected to one of the servers s_j which are called *contact servers* in [10], [20]. Contact servers are responsible for forwarding the packets to their destinations. This defines a one-to-one mapping from the clients to the servers and it is called *server allocation* in [10]. We call it an *assignment* in this paper. We use A_g to denote the assignment for some game session g. Clearly, $A_g \subset B$. An assignment is a *semi-matching* [8], [12]. A possible assignment is shown in Fig. 2 for the example in Fig. 1.

Given a desired latency bound, our goals will be to design algorithms that not only obeys this delay bound, but also keeps the number of server used as small as possible. Hence it is not necessary to find the assignment with minimum latencies. By assigning the clients cleverly, an assignment with less number of servers could be found. For example, in Fig. 2, assume that B is the central server and the delay bound is 55. Three server are used if we use the assignment shown by solid lines. But we could achieve a better assignment with 2 servers by

assigning a to B without violating the bound.

5) Latencies: Let the set of contact servers be S', the latencies for client-server and peer-to-peer architectures are defined below.

a. The client-server architecture is generally represented as a tree T with root r being the central server and which denoted as T_r. The delay of this tree $D_{cs}(T_r)$ is the maximum delay between any two clients c_i, c_j in the tree going through the root r. We also need to consider the delay from a client to itself. Let client c_i be assigned to server s_{c_i}, the latency for the client-server architecture is $D_{cs}(T_s) = 2 \times max(d_c(c_i, s_{c_i}) + d_s(s_{c_i}, r))$.
b. The peer-to-peer architecture utilizes a subnetwork $H = (S', E', w)$ containing contact servers, intermediate servers and the clients.The latency of this subnetwork $D_{p2p}(H)$ is the diameter of H, where the diameter is the longest shortest path and it can be denoted as $D_{p2p}(H) = \max(d_c(c_i, s_{c_i}) + d_s(s_{c_i}, s_{c_j}) + d_c(c_j, s_{c_j}))$.

In the peer-to-peer model, the servers only act as routers or the so-called game proxies in [13] [14]. The event synchronizations are done on the clients.

C. Server Selection Problems

We formulate *server selection problems with real-time delay constraint (SPD)* for both client-server and peer-to-peer architectures using the notations from Section II-B. Given the system model and a delay constraint μ, the goal of SPD is to find an assignment A such that the number of servers used is minimal, and

a. Client-Server (SPD-CS): the latency of the tree T rooted at r induced by the assignment A is $D_{cs}(T_r) \leq \mu$.
b. Peer-to-Peer (SPD-P2P): the latency of the subgraph H induced by the assignment A: $D_{p2p}(H) \leq \mu$.

This class of problems is proven to be *NP-hard* by the reduction from the *set-covering problem* [10].

III. Algorithms

The goal of *SPD* is to find an assignment such that the delay is less than a realtime delay constraint μ whose value μ depends on the type of the game. Additionally, we also would like that the number of servers involved is small. We first present a polynomial-time algorithm (Algorithm 1) for the *SPD-CS* problem. For a given server network topology, this algorithm will give the minimum latency (Γ) that can be achieved without any consideration to the number of servers involved. Next we present a heuristic algorithm (Algorithm 2) for the SPD-CS taking into account the latency constraint μ ($\geq \Gamma$). Algorithm 2 attempts to reduce the number of servers used while satisfying the μ. We extend Algorithm 2 to provide a solution to the SPD-P2P (Algorithm 3). All our algorithms run in polynomial-time.

A. SSP with Real-Time Delay Constraint for Client-Server Architecture (SPD-CS)

Our idea behind finding the minimum latency Γ for the SPD-CS problem is as follows. For each node $s_i \in S$, we build a shortest path tree rooted at $r = s_i$. Then we assign each client c_i to a server s_j such that $d_c(c_i, s_j) + d_s(s_j, r)$ is minimum. After all the clients are assigned as above, we can now calculate the latency of the tree rooted at s_i. After constructing all possible trees with each s_i as the root and the assignment of clients as above, we choose the tree with the minimum latency (Γ). We can construct all-pairs shortest path in $O(n^3)$ time where n is the number of servers. Considering each server s_i as the root, we need to find the assignment for each of the m clients. For a single root s_i the total-time to complete this operation will be $O(nm)$. Since we have n trees to be considered we have a time-complexity of $O(n^2m)$. Hence the total complexity of Algorithm 1 is $O(n^2m)$ with $m > n$.

Input: $G = (S, E, w)$, C, S
Output: tree T, root r, assignment A, latency d

```
Run Floyd-Warshall algorithm on G;
r = NULL, d = ∞, A = φ;
foreach s_i ∈ S do
    d_max = 0;
    foreach c_j ∈ C do
        d_u = ∞;
        foreach s_k ∈ S do
            if d_u > d_c(c_j, s_k) + d_s(s_k, s_i) then
                Assign c_j to s_k;
                d_u = d_c(c_j, s_k) + d_s(s_k, s_i);
            end
        end
        if d_max < d_u then
            d_max = d_u;
        end
    end
    if d > d_max then
        r = s_i, d = d_max;
        Make current assignment A;
    end
end
Construct tree T from r and A;
return T, r, Γ, d
```

Algorithm 1: Exact Algorithm for SPD-CS

Our Algorithm 2 is a greedy heuristic that tries to solve the the SPD-CS problem by keeping the number of servers selected to a minimal. The algorithm first starts with a single server say r. Now the tree T consists of a single node r. It assigns a set of clients to this tree T as along as the delay constraints is not violated. If all the clients are assigned to the nodes in T, then we are done. Otherwise, we will choose a server s that is a neighbor of T (a neighbor to some node in the tree) such that server s can serve as a contact server for a maximum number of unassigned clients without violating the μ constraint. Now the node s is added to the tree T. The above process continues until all clients are assigned. The complexity of Algorithm 2 is $O(n^2m)$ with $m > n$. The number of servers involved is no more than in Algorithm 1 if we set

delay constraint as the latency from Algorithm 1 (Γ).

Input: $G = (S, E, w)$, C, B and delay bound μ
Output: rooted tree T, root r, assignment A, latency d

Run Floyd-Warshall algorithm on G;
$r = NULL$, $d = \infty$;
foreach $s_i \in S$ **do**
  $d_{max} = 0$, $not\_done = false$, $found = true$, $s = NULL$;
  foreach $c_j \in C$ **do**
    if $d_c(c_j, s_i) < \mu$ **then**
      Assign c_j to s_i and update d_{max};
    else
      $not\_done = true$, $found = false$;
    end
  end
  Mark s_i as used;
  while not_done **do**
    $s = NULL, n_{max} = 0$;
    S' = neighbors of current used servers;
    foreach $t_j \in S'$ **do**
      $n_{curr} = \#$ of clients could be assigned to s_j;
      if $n_{curr} > n_{max}$ **then**
        $s = s_j, n_{max} = n_{curr}$;
      end
    end
    foreach *unassigned client* $c_j \in U$ **do**
      if $d_c(c_j, s) + d_s(s, s_i) < \mu$ **then**
        Assign c_j to s and update d_{max};
      end
    end
    if *no unsigned clients* **then**
      $found = true, not\_done = false$;
    end
  end
  if $found$ **then**
    if $d > d_{max}$ **then**
      Make current assignment A and update d_{max};
      $r = s_i$;
    end
  end
end
Construct the tree T from r and A;
return T, r, A, d_{max}

Algorithm 2: Greedy Algorithm for SPD-CS

B. SSP with Real-Time Delay Constraint for Peer-to-Peer Architecture (SPD-P2P)

We will show that the desired peer-to-peer architecture can be constructed using the tree constructed in Algorithm 1 or 2.

Suppose we have a solution (assignment) A with root S for an *SPD-CS* problem with the delay bound μ, let $s_1 \neq s_2$ be any two contact servers in A and $c_1 \neq c_2$ be two clients assigned to s_1, s_2. Then $d_c(c_1, s_1) + d_s(s_1, s) + d_c(c_2, s_2) + d_s(s_2, s) \leq \mu$. If the communication is done in a peer-to-peer fashion, then $d(c_1, c_2) = d_c(c_1, s_1) + d_s(s_1, s_2) + d_c(c_2, s_2)$. There are two cases, **(a)** s lies on one of the shortest path between s_1 and s_2 or **(b)** not. In the case of (a), $d_c(c_1, s_1) + d_s(s_1, s_2) + d_s(c_2, s_2) = d_c(c_1, s_1) + d_c(s_1, s) + d_c(c_2, s_2) + d_s(s_2, s) \leq \mu$. In the case of (b), $d_c(c_1, s_1) + d_s(s_1, s_2) + d_c(c_2, s_2) < d_c(c_1, s_1) + d_s(s_1, s) + d_c(c_2, s_2) + d_s(s_2, s) \leq \mu$. Then a *SPD-CS* solution could be used for *SPD-P2P* with the delay bound μ.

Based on this, we can convert the tree constructed by Algorithm 1 or 2 to a network for peer-to-peer communication. The delay of such subnetwork is bound by the delay of the tree constructed from Algorithm 1 or 2.

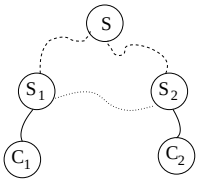

Fig. 3. Observation from an SPD-CS solution

The subnetwork construction is shown in Algorithm 3 and the idea follows. First we use Algorithm 2 to find an assignment as the solution to the SPD-P2P problem. Then for each pair of contact servers s_i, s_j, the intermediate servers between their shortest path are added to the solution. Algorithm 2 is $max(O(n^3), O(n^2 m))$. Adding intermediate servers can be done in $O(n^3)$ time if we use the all-pair shortest paths constructed earlier. The overall complexity of this algorithm is $O(n^2 m)$ with $m > n$.

Input: $G = (S, E, w)$, C, B, delay bound μ
Output: network H, assignment A and latency d

(T, A, r, d) = Algorithm 2 (G, C, B, μ);
S' = the list of used servers in A;
$H = \phi$;
$d = 0$;
foreach $s_i \in S'$ **do**
  foreach $s_j \neq s_i \in S'$ **do**
    Add s_i and s_j to H;
    Add all the servers and the edges on the shortest path between the s_i and s_j to H;
  end
end
Update the latency d;
return H, A, d

Algorithm 3: SPD-P2P Algorithm from Algorithm 2

The ZIZO algorithm in [10] which attempts to allocate the clients to the nearest servers and migrates them toward the core server s^* (that minimizes the longest shortest distance to all the clients) to reduce the number of servers used. Example shown in Fig 4 illustrates the existence of an assignment with the delay bound $\mu = 16$. However the ZIZO fails to find a solution in this example for $\mu = 18$. The initial assignment gives the minimal delay of 19 and the ZIZO algorithm stops. Our algorithms will find the solution with delay bound 18.

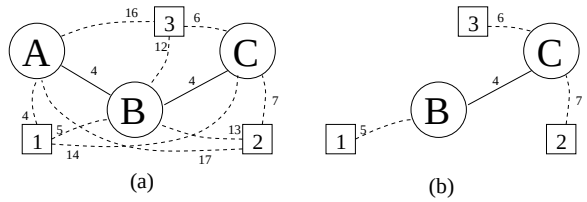

Fig. 4. (a) An example shows that ZIZO fails to find a solution with $\mu = 18$ where A, B, C are servers and 1, 2, 3 are clients. (b) A shortest path tree rooted at C has the depth of 9 which gives the delay bound $\mu = 18$.

IV. Performance Evaluation

We designed several experiments to evaluate the performance of the server selection algorithms. First we randomly generated a set of networks with $20, 25$ and 30 servers and the numbers of clients $30, 50, 80, 100$ and 120. For each combination of the number of servers and clients, 30 different networks are generated randomly with a total of 465 different networks. Then we reduce the latencies on the network of servers by 30% to represent the well-provisioned network. All graphs shown in this paper are for networks with 25 servers. The graphs for other networks are similar and removed due to the space limitation.

A. Delay Bounds

Most of the algorithms take a parameter d which is the desired delay bound for the assignments. As we mentioned earlier, the values of delay bound depend on the type of the game. However, the values must be achievable for a given network and Algorithm 1 is used to find this value Γ. Then we multiply Γ with a factor $f(\geq 1.0)$ as the delay bound. We choose $1.0, 1.1, 1.2, 1.3, 1.4$ and 1.5 as the values of f in our experiments.

B. General Experiment Method

We evaluate the performance of the algorithms based on *latency* and *number of servers involved*. The procedure we used to evaluate the algorithms follows. For each different input (netowk), Algorithm 1 is used to find the value of Γ. Note that Γ depends on the graph and hence its values are virtually unique among graphs. After Γ for a graph is found, different algorithms are used to find the assignments on this graph. This step is repeated with different delay bounds (μ) which are described earlier.

C. Evaluation of SPD-CS Algorithms

We compared the results from different algorithms for SPD-CS problems and following are the details of these algorithms.

- *Algorithm 1*: exact algorithm used to find the minimal latency (Γ) without considering the number of servers involved.
- *Algorithm 2*: our heuristic algorithm which gives a solution within the delay bound (μ) using minimal number of servers.
- *k-best*: This algorithm is used to evaluate the goodness of Algorithm 2. Suppose Algorithm 2 utilizes k servers, we search all the possible connected subgraphs with size k and run Algorithm 1 to find the least latency.

TABLE I
Algorithms compared in SPD-CS experiment

Notation	Algorithm	Complexity
alg-1	Algorithm 1	$O(n^2m)$
alg-2	Algorithm 2	$O(n^2m)$
k-Best	Based on k servers &Algorithm 1	$O(n^k \cdot n^2m)$

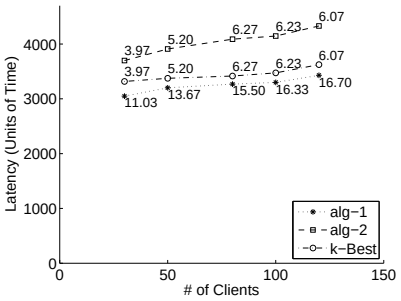

Fig. 5. Latencies on different algorithms for 25 servers, $f = 1.3$. The figure also shows the average number of servers used in the solution.

The algorithms are summarized in Table I and the results are shown in Fig. 5 and 6. The results show that all the algorithms are able to find solutions within given delay bound in different cases. We also can see that Algorithm 2 uses less number of servers in the assignments in comparison with Algorithm 1. We also found that as the delay bound increases, the number of servers used decreases (Fig. 6).

D. Evaluation of SPD-P2P Algorithms

We use a similar method and the same input to evaluate SPD-P2P algorithms. The algorithms compared are listed below.

- *Algorithm 1*: This algorithm is used in a client-server manner.
- *Algorithm 3*: Our heuristic algorithm based on Algorithm 2.
- *ZIZO*: Zoom-in Zoom-out algorithm from [10].
- *k-best*: This algorithm is similar to the SPD-CS case but Algorithm 3 is used instead of 2.

These algorithms are summarized in Table II and the results are shown in Fig. 7 and Fig. 8. The results show that our heuristic algorithm could find solutions that satisfy the delay bound (Fig. 7) given by Algorithm 1. But occasionally, *ZIZO* will not find a solution as we discussed earlier. In the comparison of number of servers, *k-Best* gives best results (fewest number of servers) while *ZIZO* is the worst (similar to alg-1(CS)). Algorithm 3 falls approximately half way between these two algorithms (Fig. 8).

TABLE II
Algorithms compared in SPD-P2P experiment

Notation	Algorithm	Complexity
alg-1	Algorithm 1	$O(n^2m)$
alg-3	Algorithm 3	$O(n^2m)$
ZIZO	Zoom-in Zoom-out	$O(nm^3)$
k-Best	Based on k servers & Algorithm 3	$O(n^k \cdot n^2m)$

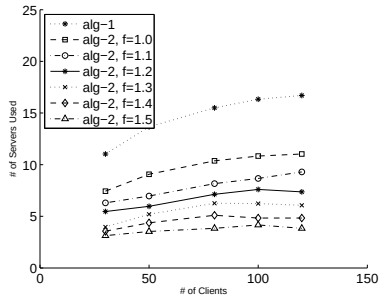

Fig. 6. Comparison on number of servers used(25 servers, $f = 1.3$).

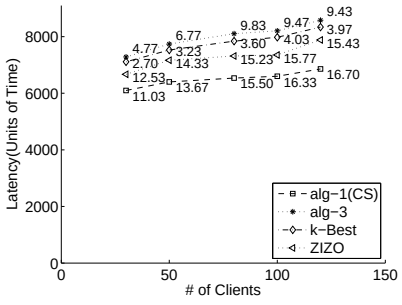

Fig. 7. Latencies on different algorithms for 25 servers, $f = 1.3$. The figure also shows the average number of servers used in the solution

V. Conclusions

Online game user experience could be improved by reducing the latencies between the users and the servers. By building a network of geographically distributed servers and connecting the servers by using high-speed low-latency links, the goal could be achieved. We designed and evaluated algorithms to minimize the number of servers used without violating delay bound for a single game session. The results show that our heuristic algorithms perform well. Although we use multiplayer online games as an example in this research, the applications are not limited to multiplayer online games. For example, networked collaborative applications such as audio/video conferences could benefit from proper server assignments. Even without using the well-provisioned network, server selection algorithms could help on finding better ways for routing for overlay networks.

In practice, these algorithms could be easily implemented since the distances matrices for the servers could be pre-calculated and this only needs to be done once. Measuring the latencies between each client-server pair could be challenging. Another issue needs to be considered is the change of group membership. Our next research topic is to do server selection for multiple groups and load balancing. We also plan to have real-world experiments on PlanetLab [16] to evaluate our algorithms.

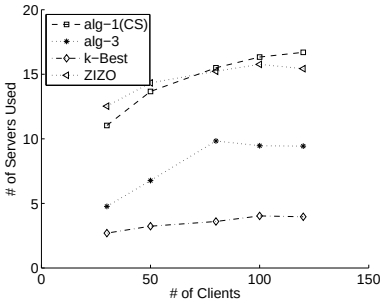

Fig. 8. Number of used servers on different algorithms($f = 1.3$)

References

[1] K.-T. Chen, C.-Y. Huang, P. Huang, and C.-L. Lei, "An empirical evaluation of tcp performance in online games," in *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2006, p. 5.

[2] K.-T. Chen, P. Huang, and C.-L. Lei, "Effect of network quality on player departure behavior in online games," *IEEE Trans. Parallel Distrib. Syst.*, vol. 20, no. 5, pp. 593–606, 2009.

[3] E. Cronin, B. Filstrup, and A. Kurc, "A distributed multiplayer game server system," in *University of Michigan*, 2001, p. 01.

[4] E. Cronin, A. R. Kurc, B. Filstrup, and S. Jamin, "An efficient synchronization mechanism for mirrored game architectures," *Multimedia Tools Appl.*, vol. 23, no. 1, pp. 7–30, 2004.

[5] S. Ferretti, "A synchronization protocol for supporting peer-to-peer multiplayer online games in overlay networks," in *DEBS '08: Proceedings of the second international conference on Distributed event-based systems*. New York, NY, USA: ACM, 2008, pp. 83–94.

[6] S. Ferretti and M. Roccetti, "Fast delivery of game events with an optimistic synchronization mechanism in massive multiplayer online games," in *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2005, pp. 405–412.

[7] F. Glinka, A. Ploss, S. Gorlatch, and J. Müller-Iden, "High-level development of multiserver online games," *Int. J. Comput. Games Technol.*, vol. 2008, pp. 1–16, 2008.

[8] N. Harvey, R. Ladner, L. Lovász, and T. Tamir, "Semi-matchings for bipartite graphs and load balancing," in *Proc. 8th WADS*, 2003, pp. 294–306.

[9] S.-Y. Hu, J.-F. Chen, and T.-H. Chen, "Von: a scalable peer-to-peer network for virtual environments," *Network, IEEE*, vol. 20, no. 4, pp. 22–31, August 2006.

[10] K.-W. Lee, B.-J. Ko, and S. Calo, "Adaptive server selection for large scale interactive online games," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM, 2004, pp. 152–157.

[11] E. Léty, T. Turletti, and F. Baccelli, "Score: a scalable communication protocol for large-scale virtual environments," *IEEE/ACM Trans. Netw.*, vol. 12, no. 2, pp. 247–260, 2004.

[12] C. P. Low, "An approximation algorithm for the load-balanced semi-matching problem in weighted bipartite graphs," *Inf. Process. Lett.*, vol. 100, no. 4, pp. 154–161, 2006.

[13] M. Mauve, S. Fischer, and J. Widmer, "A generic proxy system for networked computer games," in *NetGames '02: Proceedings of the 1st workshop on Network and system support for games*. New York, NY, USA: ACM, 2002, pp. 25–28.

[14] J. Mller, S. Fischer, S. Gorlatch, and M. Mauve, "A Proxy Server-Network for Real-time Computer Games," in *In Proc. of Euro-Par 2004*, Aug. 2004.

[15] C. Nguyen, F. Safaei, and P. Boustead, "A distributed server architecture for providing immersive audio communication to massively multiplayer online games," in *Networks, 2004. (ICON 2004). Proceedings. 12th IEEE International Conference on*, vol. 1, 16-19 2004, pp. 170 – 176 vol.1.

[16] Planetlab. [Online]. Available: https://www.planet-lab.org:443/

[17] Quake 4. [Online]. Available: http://www.quake4game.com/

[18] Starcraft. [Online]. Available: http://www.blizzard.com/us/starcraft/

[19] P. Svoboda, W. Karner, and M. Rupp, "Traffic analysis and modeling for world of warcraft," in *Communications, 2007. ICC '07. IEEE International Conference on*, 2007, pp. 1612–1617.

[20] S. D. Webb and S. Soh, "Adaptive client to mirrored-server assignment for massively multiplayer online games," in *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, ser. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, vol. 6818, Jan. 2008.

[21] World of warcraft. [Online]. Available: http://www.worldofwarcraft.com