On the game server network selection with delay and delay variation constraints

Conference Paper · February 2011

DOI: 10.1109/COMSNETS.2011.5716473 · Source: IEEE Xplore

CITATIONS
4

READS
52

4 authors:

Yuh-Rong Chen
Nanyang Technological University
6 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Sridhar Radhakrishnan
University of Oklahoma
135 PUBLICATIONS **1,055** CITATIONS
SEE PROFILE

Sudarshan Dhall
University of Oklahoma
71 PUBLICATIONS **1,758** CITATIONS
SEE PROFILE

Suleyman Karabuk
University of Oklahoma
21 PUBLICATIONS **488** CITATIONS
SEE PROFILE

On the Game Server Network Selection with Delay and Delay Variation Constraints

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—**Recent advances in multimedia software and hardware technologies and the availability of high-speed Internet service have been instrumental for growth in the online gaming industry. Multiple servers distributed across the network are commonly used to provide the desired quality-of-service (QoS) for the network game in order to achieve a higher quality-of-experience (QoE) to the players (clients). Each player in this distributed multi-player gaming environment connects to a particular server and it distributes each of the actions to all other players through the servers they are connected to. We imagine the server network to be an overlay network, wherein the latency on a link between two servers is the latency of the Internet path connecting them. We assume that we are given an overlay network of servers with link latencies and a set of players each with a different latency to each of the servers. Now our goal is to develop algorithms that perform the following actions in such a way that delay related QoS constraints are satisfied: (a) choose a subnetwork of the server network (server network selection) and (b) assign each player to a server in the subnetwork (client-assignment). More specifically, the QoS constraints that we address in this paper are a bound on the maximum delay in propagating a player's move to all other players (delay bound) and a bound on the maximum difference in the arrival times of a player's move at all other players (delay-variation bound). We have provided polynomial-time heuristics to determine a minimal cardinality server network and the corresponding client-assignment that satisfy both delay bound and that minimize delay-variation, if such a solution exists. We have considered cases in which the server network follows two communication models: client-server (CS) and peer-to-peer (P2P). Our extensive empirical studies indicate that our heuristic uses significantly less run-time in achieving the tightest delay variation for a given end-to-end delay bound while choosing a minimal number of servers.**

I. Introduction

Online games have been popular with the availability of affordable high-speed Internet service and related software and hardware technologies. The pervasive availability of game consoles with Internet capability such as Xbox, PlayStation 3 and corresponding online multiplayer gaming services like XBox Live and PlayStation Network have enabled a wide-range of individuals with none or little experience in computing to interact with others in the Networked Virtual Environments (NVEs). While many of the currently available games support a small number of players (6-32 in many cases), most of them are unable to scale [3], [4]. It has been clearly established that this lack of scalability of many of the games is directly related to network latency [6].

Researches have been done on traffic analysis and modeling [23] and user behaviors under various network quality [4] to help us understand the network requirements of different types of online games. There are also researches on proposed architectures for huge virtual environments or online games [15], [13]. It has been shown that well-provisioned networks with good server selection algorithms could be used to solve the latency problem [14], [24].

Quality-of-Experience (QoE) for online game playing is not only affected by the latencies, but also by fairness. Cross-Language Battlegroups that players from different regions of the world are selected to compete in the same game session have been implemented in the game World of Warcraft [25]. In the implementation, each instance/session of the game is handled by a single server and hence each player has a different latency to the server. The results in different levels of responsiveness for the players in the same game session, which has been shown to be unfair in competitive online games [1]. While event synchronization protocols proposed in [8] [10] are used to maintain consistent games state are important for achieving fairness, they cannot be used to solve the delay variation issue. There are solutions such as finding alternative paths or packet buffering [2].

The problems considered in this paper are closely related to the various multicast tree construction problems that have been considered in the literature [21], [22], [2]. The construction of a delay and delay-variation bounded multicasting tree (DVBMT) was first discussed and proven to be NP-complete in [22].

The main contribution of our research compared to the works on multicasting tree construction is that we consider both the configuration of the server network and the set of clients that are to be connected to the server(s), the latter of which adds an extra dimension of complexity.

The DVBMT problem is briefly defined as follows. Given a network $G = (S, E)$, a source node $s \in S$ and a set of destinations $C = \{c_1, c_2, \cdots c_n\} \subset S$. The objective is to find a set of paths to all the destinations that forms a multicast tree rooted at s such that $max(d(s, c_i) - d(s, c_j))$ (delay variation) is minimum for $c_i, c_j \in C$, where $d(s, c_i)$ denotes the delay from s to c_i for the path used. Minimizing delay-variation will improve fairness in game playing – ideally, a player close to the game server should not be able to make the moves or have a ability to know the game state any faster than other players farther away from the server.

For our problem at hand, we consider a server network whose servers are nodes in a network and link delays are the latencies of the path connecting those pair of servers nodes. In addition to the server network, we are given a set of client nodes (players of the game) with delay to each of the servers. We have designed a server subnetwork construction algorithms for either the client/server or the peer-to-peer communication model together with appropriate client to sever assignment (client assignment) in such a way that the delay related QoS requirements of the online game are met.

Minimizing the number of servers used is also one of our goals of this paper. If less number of servers are used for a single game session, it's more likely that the network of servers could accommodate more game sessions. Hence, we prefer assignments with fewest number of servers if the delay bound is satisfied. Additionally our algorithms attempt to improve the delay variation to address fairness issue, while satisfying the delay constraint.

Banik et al [2] have provided heuristic based on k-shortest path to determine client to serve path that tries to satisfy bound established for both client to server communication delay and a bound for delay variation. The difference between this work and the work in [2] are as follows:

- Banik et. al [2] considers a network containing nodes, some of which are clients and a node is selected as a server. In our paper, the nodes of the network are all routers, clients outside of the network can connect to any of the routers with each connection having a specified delay. Our goal here is to find to which nodes the clients should connect to in order to solve the problem at hand. Additionally, we have to determine the appropriate router to which the server should be connected to. If we assume that the delay from a client to any router is the shortest path delay in the network under consideration, then the work in [2] can be used to solve our problem. The network we have considered is a well-provisioned network and the path from clients to the routers in the well-provisioned network need not use the routers in this network. Hence we have considered a more general case.
- Additionally, we have addressed our problem in the context of (a) the servers could be deployed at any location of the network and (b) we have also considered the peer-to-peer architecture.

The paper is organized as follows. In Section II, we discuss the system model and problem definition/formulation. Our proposed algorithms are described in Section III. The performance of proposed algorithms is given in Section IV. Conclusions are drawn in Section V.

II. System Model and Problem Formulation

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, a *client* refers to a computer with the software installed for playing the game that renders and presents the game states to the user [14].

The network architecture for an online game dictates the mechanisms for maintaining the game states. Generally network architecture could be classified into two types according to the communication model: *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 the central server first. Then the central server computes the new state of the game and sends necessary updates to all 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 First Person Shooting (FPS) games or Massively Multiplayer Online Role Playing Games (MORPGs) such as Quake 4 and World of Warcraft use this approach. 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. We define the latency as the largest communication delay between any pair of clients. Real-time strategy (RTS) games such as StarCraft adopt this approach.

Both architectures require some synchronization mechanism to restore the order of the events. However, the 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 [9].

There are other architectures. The multiserver architecture is a variation of the client-server architecture. It is usually implemented in MMORPGs with each server being responsible for a portion (e.g., a region) of the game [11]. Mirrored server architecture is a hybrid of client-server architecture and peer-to-peer architecture [7], [10]. 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.

B. Server Network

We borrow the terminologies and notations from [14], [24] to describe Server Selection Problems, which are also used in our previous research [5].

Server network has been modeled as an overlay network [17] [18] [19] on top of the existing Internet or as a private network dedicated to the game [14]. We assume that the server

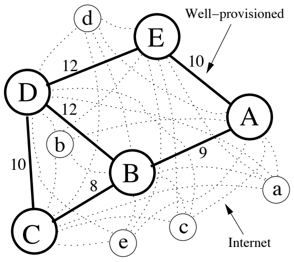

Fig. 1. An example of a game session with the set of servers $S = \{A, B, C, D, E\}$ and the set of clients $C_g = \{a, b, c, d, e\}$. The bold lines represent the well-provisioned network between the servers and the dotted lines represent Internet links between the clients and the servers.

TABLE I
LATENCIES BETWEEN CLIENTS AND SERVERS IN FIG. 1

client	a	a	a	a	a
server	A	B	C	D	E
distance	41	55	39	70	65
client	b	b	b	b	b
server	A	B	C	D	E
distance	52	32	43	25	48
client	c	c	c	c	c
server	A	B	C	D	E
distance	51	37	32	72	48
client	d	d	d	d	e
server	A	B	C	D	E
distance	52	64	60	39	48
client	e	e	e	e	e
server	A	B	C	D	E
distance	68	37	56	55	63

network is a well-provisioned network capable of providing low latencies on its links. In practice, the servers could be hosted at the Internet Service Providers in different regions to achieve this requirement. The major advantage of such a network is that the game provider will be able to use custom protocols and routing algorithms on this network. We use an undirected weighted graph $G = (S, E, w)$ to represent this network, where $S = \{s_1, ..., s_n\}$ is the set of n servers distributed geographically, E is the set of links between servers and w is the link latency function over E. An example is shown in Fig. 1 with $S = \{A, B, C, D, E\}$, $E = \{(A, B), (A, E), (B, C), (B, D), (C, D), (D, E)\}$ and the number are the link latencies between servers (w).

C. Clients

The clients are the computers used by players involved in the same game session. The number of clients in a single game session is typically between 4 and 80; examples include StarCraft II and World of Warcraft raid. Large number of game sessions are in execution at the same time and the length of each game session varies from the games. A typical World of Warcraft raid could take 3 to 4 hours with minor changes to the membership, on the other hand, a typical StarCraft II game lasts from 30 minutes to 1 hours. We assume the length of a game session is long enough so the users could benefit from the pre-arranged server communications.

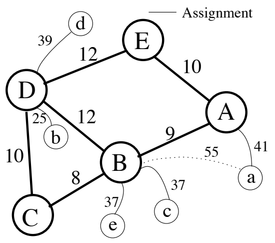

Fig. 2. An example of an assignment based on Fig. 1. The numbers denote the latencies of links.

We use $C = \{c_1, ..., c_m\}$ to denote the set of m clients (also distributed geographically) participating in the same game instance or session.

Accessing the Servers: We assume that there exists an *Internet* link between each server s_i and client c_j pair with the latency $d(c_i, s_j)$. Let $B = \{(c_i, s_j)|c_i \in C_g, s_j \in S\}$ be the set containing all these links (note: $|B| = m \times n$). In addition, disjoint vertex sets C, S and with edge set B could be visualized as a complete bipartite graph. In the example in Fig. 1, $B = \{(x, Y)|x \in \{a, b, c, d, e\}, Y \in \{A, B, C, D, E\}\}$. The latencies are given in Table I.

In the client-server model, one of the server nodes will be chosen as the central server while other servers act like routers. All the clients can connect (a) directly to the central server through an Internet path, or (b) to some other server 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 of (a) can be more than that of (b). An example is shown in Fig. 2 with B be the central server. The latency for client a is 110 (55×2) in case (a) but reduced to 100 ($(41 + 9) \times 2$) in case (b).

D. Assignments

In order to participate in a game instance, each client is assigned to one of the servers, which are called *contact servers* [14], [24]. Contact servers are responsible for forwarding the packets to their destinations. This defines a one-to-one mapping from the clients to the servers is called *server allocation* in [14]. We use the term *assignment* in this paper. We use A to denote an assignment for some game session. Clearly, $A \subset B$. Note that an assignment is a *semi-matching* [12], [16].

Given a desired latency bound, our goal is to find the sub-networks that not only obey this delay bound, but also keep the number of server used as small as possible in order to provide more simultaneous game sessions. For example, a possible assignment $A = \{(a, A), (b, D), (c, B), (d, D), (e, B)\} \subset B$ of Fig. 1 is shown in Fig. 2. Assuming that B is the central server and the delay bound is 110. Three servers 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.

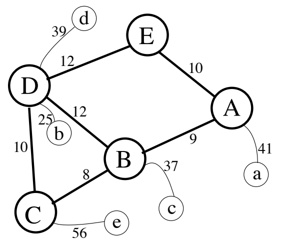

Fig. 3. Another example of an assignment based on Fig. 1. The numbers denote the latencies of links.

E. Delays (Latencies)

Let the set of contact servers be S', The latencies in a game session for both client-server and peer-to-peer architectures are defined as follows.

a. The client-server architecture is generally represented as a tree T_r with root r being the central server. The delay (or latency) 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 $D_{cs}(T_r) = 2 \times max(d(c_i, s_{c_i}) + d(s_{c_i}, r))$. Since the tree could be derived from the centralized server r and the assignment A, we use $D_{cs}(A_r)$ to denote the delay. In the previous example (Fig. 2), if $r = B$, $D_{cs}(A_r) = 2 \times (d(d, D) + d(D, B)) = 102$.
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 which can be denoted as $D_{p2p}(H) = \max(d(c_i, s_{c_i}) + d(s_{c_i}, s_{c_j}) + d(c_j, s_{c_j}))$. Similarly, we use $D_{p2p}(A)$ to denote the delay. In the same example, $D_{p2p}(A) = 101$ which is the latency between a and d using the assignment A and the path (a, A, B, D, d).

F. Delay Variation

The concept from [22] is used define delay variation for the Client-Server architecture, and we extend it for the Peer-to-Peer architecture.

a. For the client-server architecture, delay variation is the difference of the smallest latency and the largest latency among all clients: $V_{cs}(T_r) = \max |2 \times (d(c_i, s_{c_i}) + d(s_{c_i}, r)) - 2 \times (d(c_j, s_{c_j}) + d(s_{c_j}, r))|$ where $c_i, c_j \in C_g$ and $s_{c_i}, s_{c_j} \in S'$. Using the assignment in Fig. 3 with root $r = D$, the largest RTT delay is from e to D (132) while the smallest is from b to D (50). Delay variation is 82.
b. For the peer-to-peer architecture, we define delay variation as the difference of the smallest and largest latencies between two clients: $V_{p2p}(H) = \max |(d(c_i, s_{c_i}) + d(s_{c_i}, s_{c_j}) + d(c_j, s_{c_j})) - (d(c_p, s_{c_p}) + d(s_{c_p}, s_{c_q}) + d(c_q, s_{c_q}))|$ where $c_i, c_j, c_p, c_q \in C_g$ and $s_{c_i}, s_{c_j}, s_{c_p}, s_{c_q} \in S'$. In the same example (Fig. 3) with assignment A, the largest latency between two clients is 114 (a and e) and the smallest is 64 (b and d, or a and c). Delay variation is $114 - 64 = 50$.

G. Problem Formulations

We formulated *Server Selection Problems with Delay Constraints* (SPD) for both client-server and peer-to-peer architectures using the notations from Section II. 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 $D_{cs}(A_r) \leq \mu$.
b. Peer-to-Peer (SPD-P2P): the latency of the subgraph H induced by the assignment A: $D_{p2p}(A) \leq \mu$.

This class of problems is proven to be *NP-hard* by the reduction from the *set-covering problem* [14].

We further extend this by reducing the delay variation for a given assignment. *Server Selection Problems with Delay Constraints and Delay Variation Reduction* (SPDVR). It is defined as follows:

Given an assignment A satisfying the delay constraint μ, the goal of SPDVR is to find a a new assignment A' using the same set of servers in A such that the delay is less than μ and the delay variation is minimal.

Note that since $A' \subseteq A$, the number of servers used in the new network is no more than the original solution (A).

III. Algorithms

We discussed and evaluated the algorithms for *SPD* in our previous research [5]. In this section, we will first briefly introduce the ideas of our heuristic SPD algorithms in Section III-A and present SPDVR algorithms for both client-server (SPDVR-CS) and peer-to-peer architectures (SPDVR-P2P) in Section III-B and III-C, respectively. Note that all the algorithms run in polynomial time.

A. Algorithms for SPD

We briefly introduce our SPD algorithms and make comparison with *ZIZO* algorithm from [14]. Although *ZIZO* algorithm is for the *mirrored server* architecture, it could be used for SPD-P2P since it's a variation of the peer-to-peer model.

- *Algorithm Matching*: This algorithm is used to find the minimum latency Γ in a given network for the SPD-CS problems. The the number of the servers used are not considered. It works as follows. For each server $s_i \in S$, 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 $d(c_i, s_j) + d(s_j, r)$ is minimum. After all the clients are assigned as above, we calculate the latency of current assignment. After testing all possible trees, we choose the one with the minimum latency (Γ). All-pairs shortest path can be done in $O(n^3)$ time. For each s_i, the total time to find the assignment describe above is $O(nm)$, results in $O(n^2m)$ for n distinct trees. The overall complexity is $O(n^2m)$ with $m > n$.

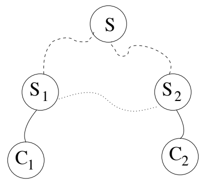

Fig. 4. Observation from an SPD-CS solution

- *Algorithm SPDCS-H*: This heuristic algorithm for *SPD-CS* takes a greedy approach by keeping the number of servers selected to a minimal. It begins with a server r (as the root of the tree T). Then a set of clients are assigned to this tree T if the delay constraint 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 the number of unassigned clients can be assigned to it without violating the delay constraint is maximum. The clients will be assigned to this server and the s is added to T. Repeat until all clients are assigned. The complexity of this algorithm is $O(n^2m)$ with $m > n$. The number of servers involved is no more than in *Algorithm Matching* if the delay constraint is greater than or equal to Γ.

We use Fig. 4 to show that the desired peer-to-peer architecture can be constructed using the tree from SPD-CS algorithms.

Suppose we have an 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, respectively. Then $d(c_1, s_1) + d(s_1, s) + d(c_2, s_2) + d(s_2, s) \leq \mu$. If the communication is done in a peer-to-peer fashion, then $d(c_1, c_2) = d(c_1, s_1) + d(s_1, s_2) + d(c_2, s_2)$. There are two cases, s lies on one of the shortest paths between s_1 and s_2 or not. In either case, $d(c_1, s_1) + d(s_1, s_2) + d(c_2, s_2) \leq d(c_1, s_1) + d(s_1, s) + d(c_2, s_2) + d(s_2, s) \leq \mu$ Hence a *SPD-CS* solution could be used for *SPD-P2P* with the delay bound μ.

Based on this observation, we can convert the tree constructed using SPD algorithms to a network for peer-to-peer communication. The delay of such subnetwork is also bound by the delay of the tree.

- *Algorithm SPDP2P-H*: This algorithm works as follows. First we use a SPD algorithm to find an assignment as the solution to the SPD-P2P problem. Then for each pair of contact servers s_i, s_j in the solution, the intermediate servers along their shortest path are added to the solution. The complexity of the SPDCS-H algorithm is $O(n^2m)$. Adding finding the intermediate servers can be done in $O(n^3)$ time. The overall complexity of this algorithm is $O(n^2m)$ with $m > n$.
- *Zoom-In Zoom-Out*: The *ZIZO* algorithm in [14] first allocates 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 5 illustrates the existence of an assignment with the delay bound $\mu = 16$. However 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 algorithm finds the solution with delay bound 16.

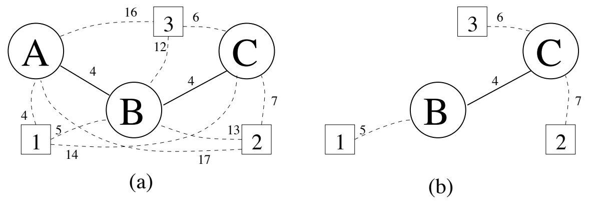

Fig. 5. (a) An example shows that ZIZO fails to find a solution with $\mu = 18$ where A, B, C are servers and 1, 2, 3 are clients. (b) A shortest path tree rooted at C has the depth of 9 which gives the delay bound $\mu = 18$. Our our heuristic will find a solution with the delay 16 if we set the delay bound parameter to 18.

B. Algorithm for SPDVR-CS

The goal of SPDVR-CS problems is similar to DVBMT (Delay and delay Variation-Bound Multicast Tree) problems [21], [22]. We borrow *Algorithm Chains* from [2] as part of Algorithm 1 to solve SPDVR-CS problems.

To find a solution with the minimal delay variation from the given assignment A, we first find the set of servers used S' (assume $|S'| = k$). Next we find the shortest distances from root r to all other nodes in S'. Each client now could use one of the servers in S' as its contact server. A list of tuples are created: $L = \{(c_i, s_j, d_{j,i}) | c_i \in C, s_j \in S'\}$ and $d_{j,i}$ is the latency from client c_i to r using contact server s_j. $(c_i, c_j, d_{j,i})$ will not be included if $d_{j,i} > \mu$.

Now we can apply *Algorithm Chain* to find an assignment with minimum delay variation. First L is sorted by $d_{j,i}$ in non-decreasing order. Let h, t be two elements in L where $h < t$. We define a *chain* be a set of consecutive elements in L such that all the clients are covered and use $T(h, t)$ to denote it where the h is the head and t is the tail. Define the delay variation of the chain as $D(h, t)$ is $d(t) - d(h)$, where $d(h)$ and $d(t)$ are the latencies of h and t, respectively. The size of a chain must be greater than or equal to $|C|$ (the number of the clients).

Now, for each element s in L as the head, scan toward the end of the list until each of the clients is visited at least once. All possible *minimal delay variation* chains can be found and the one with the smallest delay variation will be our solution. The execution of the algorithm on the example in Fig. 1 is demonstrated below.

Assume $\mu = 139$. The set of servers used in assignment A is $S' = \{A, B, C, D\}$. Then we compute the distance from the central server D to all servers in S': $d(A, D) = 21, d(B, D) = 12, d(C, D) = 10$. Next is to construct the list L and sort L by d_{ji} in non-descending order and apply *Algorithm Chains*. If $d_{ji} > \frac{\mu}{2}$, the delay bound is violated and the tuple will not be added to L.

The sorted list L can be constructed from Fig. 1 and I. Fig. 6 shows the idea of constructing L. Starting with the centralized server D and find the shortest path to servers used

TABLE II
SORTED LIST L

id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
i	b	c	b	c	e	d	a	a	b	c	b	d	e	a	e	d	d	e
j	D	C	B	B	B	D	C	A	C	A	A	A	D	B	C	C	B	A
d_j	25	32	32	37	37	39	39	41	43	51	52	52	55	55	56	60	64	68

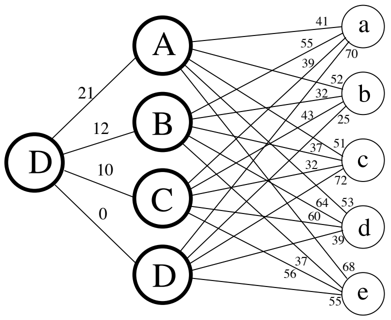

Fig. 6. The temporary structure for constructing L.

(A, B, C, D). The numbers on the links denote the distances between two servers. Then we consider the delays between clients a, b, c, d, e and servers A, B, C, D. By adding them together, we can find the delays from a client to the centralized server D through a possible contact server. For example, the distance between client b to server D through contact server C is $43 + 10 = 53$. The set of tuples $\{(c_i, s_j, d_{j,i}) | c_i \in C, s_j \in \{A, B, C, D\}\}$ is created and the sorted. L is shown in Table II.

For each element in L, we attempt to find a chain if it exists. For example, a chain starting with the first element ($id = 1$) is denoted as $H_1 = \{1, 2, 3, 4, 5, 6, 7\}$ covers all the clients. The delay variation of H_1 is $D(1, 7) = 39 - 25 = 14$. A chain starts with h is minimal if it is no longer a chain if we remove the last element in the chain. We listed the delay variation of all possible minimal chains (Table III) for the graph in Fig. 6. From the table, the chain $10, 11, 12, 13, 14$ has the smallest delay variation and is chosen as our solution.

The algorithm (*SPDVRCS-H*) is shown in Algorithm 1. The set of servers used can be found in $O(m)$ time (line 1). In line 2-9, creating and sorting the list L cost $O(mn)$ time. Line 11-17 is the main part of *Algorithm Chains*. For each element in L, the scan costs at most $O(mn)$ of time. In the worst case, it takes $O(m^2n^2)$ of time. The overall time complexity of Algorithm 1 is $O(m^2n^2)$.

TABLE III
ALL POSSIBLE CHAINS FROM L

$Chain$	$d(h)$	$d(t)$	$D(h,t)$
$1, 2, 3, 4, 5, 6, 7$	25	39	14
$2, 3, 4, 5, 6, 7$	32	39	7
$3, 4, 5, 6, 7$	32	39	7
$4, 5, 6, 7, 8, 9$	37	43	6
$5, 6, 7, 8, 9, 10$	37	51	14
$6, 7, 8, 9, 10, 11, 12, 13$	39	55	16
$7, 8, 9, 10, 11, 12, 13$	39	55	16
$8, 9, 10, 11, 12, 13$	41	55	14
$9, 10, 11, 12, 13, 14$	43	55	12
$10, 11, 12, 13, 14$	51	55	4

```
   Input: $G = (S, E, w)$, $C$, $B$, root $r$, assignment $A$,
          delay bound $\mu$
   Output: $dv$, new assignment $A'$
1  Create the set of servers involved in $A$ as $S'$;
2  Create an empty list $L$;
3  foreach $c_i \in C, s_j \in S'$ pair do
4      Compute the $D_{c_i,s_j} = d_c(c_i, s_j) + d_s(s_j, r)$;
5      if $D_{c_i,s_j} < \mu$ then
6          Add the tuple $(c_i, s_j, D_{c_i,s_j})$ to $L$;
7      end
8  end
9  Sort $L$ by $D_{c_i,s_j}$ in non-descending order;
10 $dv = \infty$;
11 foreach $T_i \in L$ do
12     Scan from $T_i$ to the right until all $c_i$ are visited,
       let the last one be $T_k = (c_m, s_n, D_{c_m,s_n})$;
13     if $D_{c_m,s_n} - D_{c_i,s_j} < \delta$ then
14         $dv = D_{c_m,s_n} - D_{c_i,s_j}$;
15         Create a new assignment $A'$ from $T_i$;
16     end
17 end
18 return $dv$, $A'$
```

Algorithm 1: Heuristic Algorithm for SPDVR-CS

C. Algorithm for SPDVR-P2P

The delay variation for the Peer-to-Peer architecture is the difference between the maximum and minimum delay between any two clients. To reduce the delay variation, we could either try to decrease the maximum delay or increase the minimum delay by modifying current assignment. However, it cannot be guaranteed that the delay variation is reduced since all the latencies have to be recalculated after the reassignment. We will demonstrate this using the example in Fig 3.

As we have shown in Section II-F, the delay variation for the Peer-to-Peer architecture is 50. One way to reduce the delay variation smaller is to make the smallest latency (64 between b and d) larger. In this case, we could reassign client d to server A and the latency between client b and client d becomes 98. However, the latencies between client d and c, a, e are also changed to 98, 93, 125 respectively. The smallest latency becomes 74 which is the latency between b and c. The delay variation becomes $125 - 74 = 51$ which becomes larger. This example shows that reducing the delay variation in the Peer-to-Peer architecture is more difficult than the Client-Server architecture.

The heuristic algorithm (Algorithm 2, *SPDVRP2P-H*) we proposed is based on *client reassignment* and works as follows. For a given assignment A, we find the set of servers involved in the assignment S'. A list L sorted by $d_{i,j}$ which contains the set of tuples $(c_i, c_j, d_{i,j})$ is created.

```
    Input: $G = (S, E, w)$, $C$, $B$, assignment $A$, delay
           bound $\mu$
    Output: $h$, $d$, $dv$, $\Gamma$
 1  Create the set of servers involved in $A$ as $S'$;
 2  $L = ConstructL(G, C, B, A)$;
 3  $done = false$;
 4  while !$done$ do
 5      Let $c_i, c_j, d$ be the two clients and the delay in
        the first element in $L$;
 6      $(A, L, reassigned) =$
        $Reassign(c_i, G, C, B, \mu, A, S', L)$;
 7      if $reassigned == false$ then
 8          foreach $s_k \in S$ do
 9              $(A, L, reassigned) =$
                $Reassign(c_j, G, C, B, \mu, A, S', L)$;
10          end
11      end
12      if $reassigned == false$ then
13          $done = true$;
14      end
15  end
16  $done = false$;
17  while !$done$ do
18      Let $c_i, c_j, d$ be the two clients and the delay in
        the last element in $L$;
19      $(A, L, reassigned) =$
        $Reassign(c_i, G, C, B, \mu, A, S', L)$;
20      if $reassigned == false$ then
21          foreach $s_k \in S$ do
22              $(A, L, reassigned) =$
                $Reassign(c_j, G, C, B, \mu, A, S', L)$;
23          end
24      end
25      if $reassigned == false$ then
26          $done = true$;
27      end
28  end
29  return $d_v(L)$, $A$
```
Algorithm 2: Heuristic Algorithm for SPDVR-P2P

Then the algorithm goes through two phases. The delay variation of L is the difference between the delays of the first and the last elements. In the first phase, the goal is to reduce the delay variation by increasing the smallest delay in the list L. This is done by reassigning one of the two clients at the beginning of L to servers in S' by calling *Procedure Reassign()*. A new list with minimal delay variation is returned from the procedure if it exists. If a new assignment is found, A and L are replaced by A' and L respecively and repeat this phase with the new assignment. If no suitable assignment is found, then we call *Procedure Reassign()* to find a new assignment by reassigning the second client. Repeat this until no more reassignment can be found for both clients at the front of L.

```
    Input: $G$, $C$, $B$, $\mu$, $A$
    Output: $L$
 1  Construct an empty list $L$;
 2  foreach $c_i, c_j$ pair( $c_i \neq c_j$) do
 3      Compute the delay $D_{c_i,c_j}$ under current
        assignment $A$;
 4      add the tuple $(c_i, c_j, d(c_i, c_j))$ to $L$;
 5  end
 6  Sort $L$ by $d(c_i, c_j)$ in non-descending order;
 7  return $L$
```
Procedure ConstructL(G, C, B, A)

```
    Input: $c_i$, $G$, $C$, $B$, $\mu$, $A$, $L$
    Output: rooted tree $T$, root $r$, assignment $A$, latency
            $d$
 1  foreach $s_k \in S'$ do
 2      Create assignment $A'$ by reassigning $c_i$ to $s_k$;
 3      $L' = ConstructL(G, C, B, A')$;
 4      if $d_v(L') < d_v(L)$ and $d(L') \leq mu$ then
 5          Reassign $c_i$ to $s_k$ (update $A$);
 6          $L = L'$;
 7          $reassigned = true$;
 8      end
 9  end
10  return $A, L, reassigned$
```
Procedure Reassign($c_i, G, C, B, \mu, A, S', L$)

Next phase is similar but instead of reassigning the clients at the beginning of L, the clients at the end of L are reassigned to reduce the largest delay. We use the previous example (Fig. 1 and Table I) to demonstrate this algorithm.

In the example, $S' = \{A, B, C, D\}$. First we construct the sorted list L (Table IV) which represents the latencies between all pairs of clients. The current delay variation is $114(a, e) - 64(b, d) = 50$. Now we start the first phase with client b which is currently assigned to server D. The elements $(a, b), (b, c), (b, d), (b, e)$ in L are affected after reassigning b to A, B or C, and the new latency values are shown in Table V. We can see that if we reassign client b to server C, the delay variation is improved and also the smallest. Hence client b is reassigned to server C and L is updated. This is repeated until the delay variation cannot be reduced. Then the algorithm enters the second phase to reduce the delay variation from the end of the list. The algorithm stops and returns the current assignment. For this example, the assignment

TABLE IV
INITIAL SORTED LIST FOR SPDVR EXAMPLE

Client Pair	b, d	b, c	a, b	a, c	c, d
Latency	64	74	87	87	88
Client Pair	b, e	a, d	c, e	d, e	a, e
Latency	91	101	101	105	114

TABLE V
RESULTS AFTER REASSIGN CLIENT b

	New Latencies for Client Pairs			
New Server for b	a, b	b, c	b, d	b, e
A	93	98	112	125
B	82	69	83	96
C	101	88	92	99

$(a, B), (b, C), (c, B), (d, D), (e, C)$ with delay variation 18 is found.

SPDVRP2P-H is shown in Algorithm 2. The set of used servers S' and L constructed in lines 1-2 can be done in $O(m^2)$ if all-pair shortest path between servers are given. A single iteration in the first loop (lines 4-15) is dominated by *Procedure Reassign()* which can be done in $O(nm^2)$. The while loop in lines 17-28 takes $O(nm^2)$ time for a single iteration. Define the delay variation of the given assignment as D_v. These two loops are executed at most D_v times which could not be determined before L was constructed (D_v is also bound by μ). Hence the algorithm has pseudo-polynomial complexity $O(nm^2)$. In the simulations, the first loop (phase) is only executed a few times (< 10) and the second phase even fewer.

IV. PERFORMANCE EVALUATION

Experiments are designed to evaluate the performance of our algorithms. First a set of networks with $20, 25$ and 30 servers and $30, 50, 80, 100$ and 120 clients are randomly generated. For each combination, there are 30 different networks with 450 different networks in total. The latencies on the network of servers are reduced by 30% to represent the well-provisioned network. All graphs shown in this paper are for networks contains 25 servers. The graphs for other networks are similar but omitted 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, it must be achievable on a given network and we use *Algorithm Matching* to find this value Γ, which is the latency of the shortest shortest-path tree. Then we multiply Γ with a factor $f(\geq 1.0)$ as the delay bound μ. In the experiments, we scaled Γ to 500 ms and we choose $1.0, 1.1, 1.2, 1.3, 1.4$ and 1.5 for f in our experiments.

B. General Experiment Method

We evaluated the performance of the algorithms based on *delay variation reduced* after applying our algorithms. The procedure follows. For each different input (network), we first find the best possible value of Γ. After Γ for a graph is found, different SPD algorithms are used to find the initial assignments satisfying μ. Repeat with different delay bounds (μ) as described earlier. The evaluation of SPD algorithms is shown in Section IV-C. Then we apply our delay variation reduction algorithm to reduce the delay variation. The delay variation of the initial and final assignments are compared.

TABLE VI
ALGORITHMS COMPARED IN SPD-CS EXPERIMENT

Notation	Algorithm	Complexity
alg-m	Algorithm Matching	$O(n^2m)$
spdcs-h	Algorithm SPDCS-H	$O(n^2m)$
k-best	Best result utilizing k servers & Algorithm Matching (k depends on results of Algorithm SPDCS-H)	$O(n^k \cdot n^2m)$

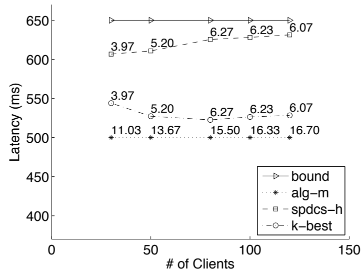

Fig. 7. Latencies on different algorithms for 25 servers, $f = 1.3$. The figure also shows the average number of servers used in the solution.

C. Evaluation of SPD Algorithms

To evaluate our results in terms of the number of servers used in the feasible solution, we generated all subnetworks of size k for the given network where k is given by *SPDCS-H*. For all subnetworks of size k that admits a feasible solution, we chose one that has the least delay and delay variation values. We call this *k-best* algorithm.

1) SPD-CS Algorithms: We compared the results from different SPD-CS algorithms in Table VI. The results show that all the algorithms are able to find solutions within given delay bound. *Algorithm SPDCS-H* found assignments with fewer servers than *Algorithm Matching* (Fig 7). We also observed that as the delay bound increases, the number of servers used decreases.

2) SPD-P2P Algorithms: The same method and input are used to evaluate SPD-P2P algorithms in Table VII. Our heuristic algorithms are able to find solutions that satisfy the delay bound (Fig. 8). But occasionally, *ZIZO* can not find a solution. On the number of servers used, *k-best* gives best results (fewest number of servers) while *ZIZO* is the worst (similar to alg-m(CS)). *Algorithm SPDP2P-H* falls approximately half way between these two algorithms (Fig. 9).

TABLE VII
ALGORITHMS COMPARED IN SPD-P2P EXPERIMENT

Notation	Algorithm	Complexity
alg-mp	Algorithm Matching (converted to P2P)	$O(n^2m)$
spdp2p-h	Algorithm SPDP2P-H	$O(n^2m)$
zizo	Zoom-in Zoom-out	$O(nm^3)$
k-best	Best result utilizing k servers & Algorithm SPDP2P-H (k depends on results of Algorithm SPDP2P-H)	$O(n^k \cdot n^2m)$

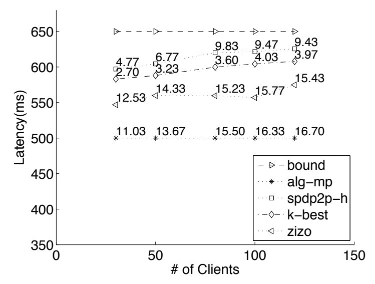

Fig. 8. Latencies on different algorithms for 25 servers, $f = 1.3$. The figure also shows the average number of servers used in the solution

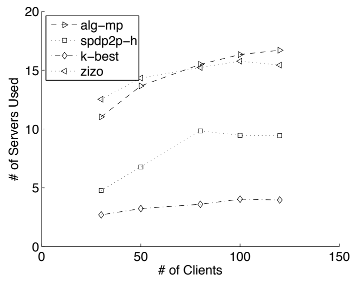

Fig. 9. Number of used servers on different algorithms($f = 1.3$)

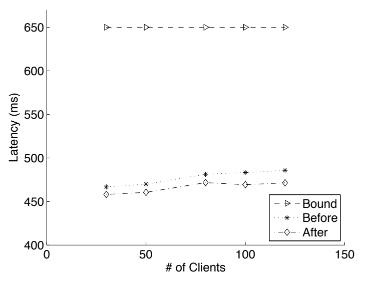

Fig. 11. Delay before and after applying Algorithms 1 (25 servers, $f = 1.3$)

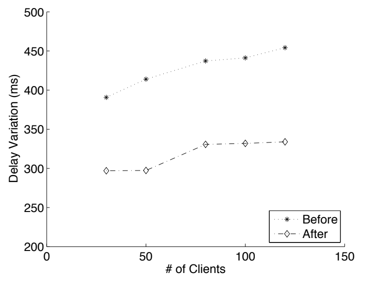

Fig. 12. Delay variations before and after applying Algorithms 2 (25 servers, $f = 1.3$)

D. Evaluation of SPDVR-CS Algorithm

We compare the improvement on delay variation after Algorithm 1 (SPDVRCS-H) is applied. Algorithm 1 significantly reduces the delay variation on different graphs. The result of the networks with 25 *servers* $f = 1.3$ is shown in Fig 10 while other configurations give similar results.

We also compared the change on latencies after applying Algorithm 1. Interestingly, we found that Algorithm 1 does not only reduce the delay variation, but it also reduces the latencies by a small amount (Fig 11). Similar results are observed for other input data sets.

E. Evaluation of SPDVR-P2P Algorithm

Similar comparisons are done for Algorithm 2. The results show that Algorithm 2 could reduce the delay variation (Fig 12). The results for other configurations are similar.

On the latencies, the latencies are also reduced as shown in Fig 13 in most cases. In some of the cases ($f = 1.4, f = 1.5$), the latencies increase by a small amount after applying Algorithm 2, but they are still below the delay bound (Fig 14).

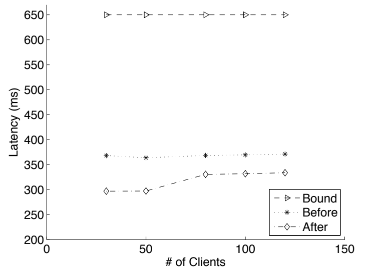

Fig. 13. Delay before and after applying Algorithms 2 (25 servers, $f = 1.3$)

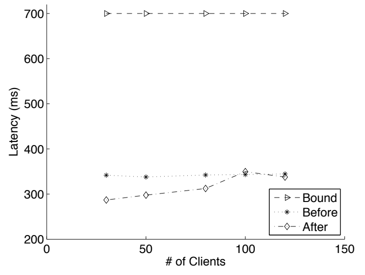

Fig. 14. Delay before and after applying Algorithms 2 (25 servers, $f = 1.4$). We observed that the latencies increase by a small amount in some cases (100 clients).

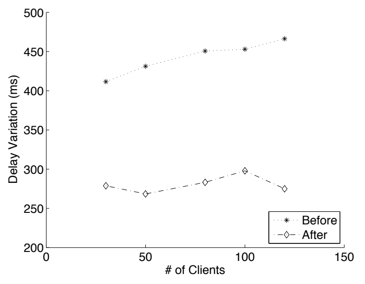

Fig. 10. Delay variations before and after applying Algorithms 1 (25 servers, $f = 1.3$)

V. Conclusions

User experience in online games can be improved by reducing user to server, and server to server connection latencies. This goal can be achieved by building a network of geographically distributed servers and connecting them with well-provisioned links.

In this paper, we first review server selection algorithms with delay constraints which also minimize the number of servers used. Next, we extend the problem by considering fairness in the online game server selection problems (delay variation). We propose two algorithms to reduce the delay variation: one for the client-server architecture and the other for the peer-to-peer architecture. Experimental results show that both heuristics effectively reduce the delay variation by reassigning the clients to different servers without increasing the number of servers used in the given assignment. Unexpectedly, we observed that both algorithms also reduce the average latencies by a small amount.

Although we use multiplayer online games as an example in this research, the applications are not limited to this particular problem domain. For example, networked collaborative applications such as distributed online music concerts, mobile gaming or audio/video conferences could benefit from proper server assignments. Even without using a well-provisioned network, server selection algorithms can help us find better ways for routing in overlay networks.

In practice, these algorithms are easy to implement due to the low computational overhead. Although the servers could be placed at the ISPs to achieve the network requirements, the choice of locations to deploy the server nodes could be a major challenge. Additionally, the choice of a appropriate overlay server network topology have be be studied in delay. This topology should take into account the various game and client requirements.

Another issue that needs to be solved is the change of group membership; in this case we would have to reallocate some of the clients to different servers. A handshaking protocol is necessary for this procedure. Our ongoing research topics are server selection problem for multiple groups and load balancing. We also plan to perform real-world experiments on PlanetLab [20] to evaluate our algorithms.

References

[1] G. Armitage, "An experimental estimation of latency sensitivity in multiplayer quake 3," in *Networks, 2003. ICON2003. The 11th IEEE International Conference on*, 2003.

[2] S. M. Banik, S. Radhakrishnan, and C. N. Sekharan, "Multicast routing with delay and delay variation constraints for collaborative applications on overlay networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 3, pp. 421–431, 2007.

[3] 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.

[4] 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.

[5] Y.-R. Chen, S. Radhakrishnan, S. Dhall, and S. Karabuk, "Server selection with delay constraints for online games," in *IEEE Globecom 2010 Workshop on Multimedia Communications and Services (MCS 2010)*, Miami, Florida, USA, 2010.

[6] M. Claypool and K. Claypool, "Latency and player actions in online games," *Commun. ACM*, vol. 49, no. 11, pp. 40–45, 2006.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] PlanetLab. https://www.planet-lab.org:443/.

[21] G. N. Rouskas and I. Baldine, "Multicast routing with end-to-end delay and delay variation constraints," *IEEE Journal on Selected Areas in Communications*, vol. 15, pp. 346–356, 1995.

[22] G. Rouskas and I. Baldine, "Multicast routing with end-to-end delay and delay variation constraints," *Selected Areas in Communications, IEEE Journal on*, vol. 15, no. 3, pp. 346–356, Apr 1997.

[23] 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.

[24] 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.

[25] World of Warcraft - Cross-Language Battlegroups FAQ. http://forums.wow-europe.com/thread.html?topicId=9446244539.