Server Selection and Topology Control for Multi-Party Video Conferences

Shuopeng Zhang[1] Di Niu[2] Yaochen Hu[2] Fangming Liu[3]
[1]University of Waterloo, Canada
[2]University of Alberta, Canada
[3]Huazhong University of Science and Technology, China

ABSTRACT

This paper proposes new methods for multi-server placement and topology control in multi-party video conferences. Given a large server pool available from CDN infrastructures and datacenter networks, our lightweight methods can rapidly determine the network topology and select the best physical servers to deploy virtualized server instances on, with the objective of minimizing the mean end-to-end delay between clients. We propose D-Grouping, a ping-based clustering algorithm, which is used in combination with convex optimization to determine the network topology and fine-tune server selection. To verify the proposed methods, we present extensive simulation studies based on the ping traces collected from 518 PlanetLab nodes as well as real-world experiment results based on a prototype implementation.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network communications; C.2.4 [**Distributed Systems**]: Client/server

General Terms

Algorithms, Design, Experimentation, Measurement, Performance

Keywords

Video Conference, Topology Management, Measurement, Latency, Internet Telephony, Network Coordinate System, Performance Optimization

1. INTRODUCTION

Multi-party conferencing, e.g., Google Hangouts, Skype group video call, is an important real-time application that enables geographically distributed clients to communicate with each other. Given the stringent requirements on quality of service (QoS), to support live interaction in such applications, the data stream of each client needs to be transmitted to all other clients with low end-to-end delays, at a reasonably high source rate.

Existing multi-party conferencing solutions mainly adopt two types of architectures, namely peer-to-peer (P2P) and centralized servers. In a typical P2P conferencing system of n clients, each client needs to send a copy of its data stream to $n-1$ other participants, making the outgoing link highly congested. A centralized-server approach relieves the burden of client outgoing links by gathering all client's data using a server called multipoint control unit (MCU) [6], and letting the server streams the processed data to other clients. Although the use of centralized servers potentially increases throughput, it may compromise delay performance as compared to a full-mesh P2P topology with direct connections between clients, especially when clients are geographically spread out.

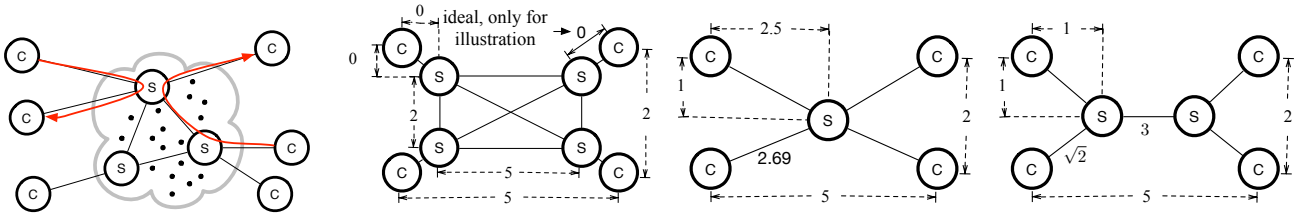
To reduce end-to-end latencies while supporting reasonably high source rates, in this paper, we propose to utilize multiple servers that are available from the large cloud of proprietary (e.g., Skype can use Microsoft servers) or third-party CDN nodes and datacenters. To shift upload burden away from clients, we can always let each client upload its data stream to some server in the cloud. Since server nodes are usually much better provisioned than hosts in residential networks, we let the servers form a full-mesh to minimize the server-to-server latency.

The key question is — given the client locations and a server number constraint, where the servers should be placed and to which server each client should upload its stream in order to reduce the overall end-to-end latencies between clients? This hard combinatorial problem involves both assignment and placement (which interact with each other), and is apparently impractical to solve on the dense graph formed by the clients and hundreds or thousands of available server nodes. An alternative solution is to embed all the nodes into a delay space using a network coordinate system like Vivaldi [2] based on pairwise pings of all servers and clients. Given the computed coordinates of all nodes, we can conveniently partition the clients using heuristics like *k-means*, and then compute the optimal server locations in the delay space under the found partition. However, as the server pool scales to a large size, there is clearly a high overhead of computing the coordinates of all nodes even with Vivaldi.

In this paper, we propose a lightweight practical solution to the problem mentioned above only based on the RTTs between the few clients and some geo-information. We propose *D-Grouping*, a novel clustering algorithm that only uses

NOSSDAV '14, March 19 - 21 2014, Singapore, Singapore.

http://dx.doi.org/10.1145/2578260.2578261



(a) Our multi-server topologies (b) The full-mesh best solution (c) A 1-server solution (d) A 2-server solution

Figure 1: An illustration of our (idealized) multi-server topology, where multiple servers are chosen from the cloud to serve each session. C: client; S: server; Arrows: data flow paths between clients.

the pairwise pings between clients to partition them, which unlike k-means, does not rely on any projected coordinates or locations. Based on the resulted partition and topology, we further use various convex optimization schemes to fine-tune the ideal server location for each group to minimize the total length of all end-to-end paths in the topology, and map ideal server locations to real servers based on certain criteria. In spite of the wide belief that geo-distances are only weakly correlated to Internet latencies [2,4], our experimental results show the surprising result that our simple lightweight scheme can greatly reduce the mean end-to-end delay over one-server solutions and even achieve a comparable performance to the heavy-weight solution based on network coordinates.

We perform extensive simulations based on the ping traces that we have collected from 518 PlanetLab nodes over a 20-day period. We have also implemented a prototype multi-party conferencing system based on *Apache Thrift* in 7,000 lines of C++ code and deployed the prototype on PlanetLab nodes to evaluate our algorithms in the real world, as well as to study the intricate relationship between source sending rates and packet-level latencies. Our packet-level experiments reveal that not only do our multi-server approach outperform the single-server solution, but its latency performance is also less sensitive to source rate increases than that of the single server, with processing overhead considered.

2. A MULTI-SERVER TOPOLOGY

We adopt a class of multi-server topologies illustrated in Fig. 1(a) to reduce delay while supporting reasonably high throughput. In this topology, every client is only connected to one server and no other host. The servers form a full mesh. Each client just sends one copy of its data stream to its own server, shifting upload bottleneck away from clients. For each server S, if it receives data from a client C, the data is forwarded to all the other clients and other servers connected to itself (server S). If server S receives data from other servers, the data is forwarded only to the clients directly connected to S. In other words, a client connected to S transmits a packet to another client connected to S in two hops via S, and transmits a packet to another client connected to another server S' in three hops via S and S'. We let the servers form a full mesh because servers are usually well connected in content delivery infrastructures or datacenter networks, which also minimizes the transfer latencies between any pair of servers.

We illustrate the benefit of multiple servers in terms of delay in Fig. 1(b), Fig. 1(c) and Fig. 1(d) in an idealized toy example of a 4-client conferencing session. If we use only one server, the mean end-to-end delay is 5.39 (omitting the unit for illustration purpose only). On the other extreme, if we place 4 servers, each close to a client and let the servers form a full mesh, the mean end-to-end delay is minimized to 4.13. To strike a balance between the mean delay achieved and the number of servers used (which is directly related to the cost), Fig. 1(d) illustrates a solution that achieves a mean end-to-end delay of 4.83 with only 2 servers. By adjusting the choice of server locations, we can effectively reduce the end-to-end delays between all clients, without causing any upload bottlenecks at the clients.

3. ALGORITHMS

Given a set of geographically distributed clients and a server number constraint m, in our protocol, we need to decide 1) where the m (virtualized) servers should be placed and 2) to which server each client connects. Since servers are directly connected, the above is no different from finding 1) an m-partition of clients and 2) the m server locations for the m partitions, respectively. We propose a three-step procedure to minimize the mean end-to-end delay between all pairs of clients. First, we propose D-Grouping (delay-based grouping) to cluster the clients only using pings between the few clients. Given the computed partition, we use convex optimization to find the ideal geographic server locations that minimize the total length of all client-to-client geographic paths in the topology formed by clients and servers. Finally, we map each ideal server location to one of the several closest physical server candidates that really achieves the minimum mean end-to-end delay.

3.1 D-Grouping based on `PINGs`

Unlike k-means which uses client coordinates (or positions) to partition them, we partition clients using the pairwise round-trip times (RTTs) between them, which can be easily retrieved by `PING` before the session starts. On the other hand, just like k-means, given the desired number of groups, D-Grouping aims to put clients with a low pairwise RTT into a same group. The intuition is that if we group "close" clients together, more traffic can be handled locally within each group, with the hope of reducing the mean end-to-end delay.

Suppose that $C = \{c_1, ..., c_n\}$ is a set of n clients. Denote $G = \{G_1, \ldots, G_m\}$ as the m groups to be computed, where each G_i is a subset of C and every client in C belongs to exactly one $G_i \in G$. D-Grouping has two steps, namely initial grouping and iterative grouping.

Initial Grouping: suppose we are given the pair-wise pings of clients and m empty groups. In initial grouping, we assign each client into the group for which it has the lowest RTT to the *polar* in that group, where a polar is a normal client used as a reference point. Suppose the number of groups m is greater than one, we first set the pair of clients with the largest RTT as two polars. Then, if m is greater than two, we will choose the non-polar client that is furthest away from the existing polars as the next polar, i.e., the client that has the largest sum of RTTs to existing polars. The above is repeated until the number of polars generated equals to the number of groups. Once polars are determined, we classify each non-polar client into the polar's group to which the client has the lowest RTT.

Iterative Grouping: next, we iteratively adjust each client into the "closest" group such that it has the minimum *average* RTT to the clients in that group, as described in Algorithm 1.

Algorithm 1 D-Grouping

while termination condition is not met, **do**
  For each client $c_i \in C$, move c_i into a group such that c_i has the minimum average delay to the clients in that group. ▷ Complexity $\mathcal{O}(n^2)$
end while

The above algorithm is lightweight with each iteration only involving $\mathcal{O}(n^2)$ evaluations of RTTs which can be conveniently collected before the session starts. In practice, the termination condition is met when a certain number of iterations T is reached or when grouping result no longer changes. In simulation, D-Grouping can yield stable partition of 12 clients in only 5 iterations for 92% of the trials.

3.2 Server Location Optimization

Once proper grouping is done, we need to choose m servers for the m groups, which is a challenging problem given the large graph of available servers. An alternative is to obtain the coordinates of all the servers and clients in a delay space, and perform a convex optimization to search for the ideal server locations in the delay space. However, embedding the hosts still involves significant overhead. Considering the correlation between geographic distance and network delay [2, 7], we propose several geo-based schemes to search for ideal server locations, which turn out to even have comparable performance with those done in a delay space.

Geo-Center: choose the geographic center of each client group as the ideal server location for that group.

Local Convex Optimization: in each group, the ideal server location is chosen to minimize the sum of geographic distances to all the clients in that group. Let $X = \{x_1, ..., x_n\}$ be the geo-locations of n clients. In each group G_i, the ideal server location L_i^* is

$$L_i^* = \arg\min_{L_i} \sum_{x_j \in G_i} D_g(L_i, x_j),$$

where D_g represents the geographic distance between two locations on the earth.

Global Convex Optimization: we use convex optimization to find all the m ideal server locations $L^* = \{L_1^*, \ldots, L_m^*\}$ that jointly minimizes the total geographic length of all end-to-end paths between clients, i.e.,

$$(n-1) \sum_{G_i \in G} \sum_{x_j \in G_i} D_g(L_i, x_j) + \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} D_g(L_i, L_j)|G_i| \cdot |G_j|,$$

where $|G_i|$ is the number of clients in group G_i.

Finally, the ideal server locations are mapped to real servers using one of the following methods:

Naive Server Search (NaiSS): choose the server geographically closest to L_i as the server of group G_i.

Local Server Search (LclSS): For each group G_i, choose p servers geographically closest to L_i as candidate servers. Then choose the server that has the smallest sum of RTTs to all the clients within group G_i as the server for G_i. To measure RTTs, the pair-wise pings between each candidate server to all the clients in its group should be performed. Thus, pn pings are performed in total. If the pings are performed in parallel by clients, only p pings need to be performed per client.

Global Server Search (GlbSS): For each group G_i, choose p servers geographically closest to L_i as candidate servers. Then choose the set of m servers from all p^m combinations of candidate servers that minimizes the mean end-to-end delay. In addition to the pings collected in Local Server Search, now we also need to collect the pair-wise pings between candidate servers from different groups, that is $mp(m-1)p/2$ pings, or $(m-1)p$ pings performed per candidate server in parallel.

4. TRACE-DRIVEN SIMULATIONS

We provide simulation results based on a large set of ping traces to evaluate our methods, compared against the state-of-the-art one-server solution and a method with much larger overhead in a delay space assuming network coordinate embedding is available through Vivaldi. We also draw insights on how many server should be used in a conference session.

We continuously collected the pair-wise pings of 518 PlanetLab nodes during a 20-day period. with the geographic distribution of the nodes shown in Fig. 2. The OS of each node is either Fedora 8 (Linux 2.6.32-20) or Fedora 14 (Linux 2.6.32-36). Everyday each node pinged all other nodes for 50 times. In total, we have collected 15.9 GB traces. In the simulations, we choose the median ping of each pair of nodes as the delay estimate of the pair, and the end-to-end delay on a certain path is calculated by summing up the RTTs of all the edges on it divided by 2.

Note that the mean end-to-end delay of different clients may have a huge difference depending on the geographic distribution of clients. For example, the mean end-to-end delay of four clients with 2 in Asia and 2 in North America is much larger than that of 4 clients all in North America. Therefore, to evaluate a scheme's performance, we use the ratio of the mean end-to-end delay of this scheme over that of the full-mesh direct connection, which we refer to as *performance ratio*). We use $p = 3$ candidate servers for Local Server Search and Global Server Search. For a given number n of clients, we run 1000 independent simulations for each method, each randomly choosing n clients from 518 nodes (the unchosen nodes will act as the available server pool), and obtain the average performance for each method.

Fig. 3 shows the performance of different combinations of proposed algorithms for 12 clients. *Firstly*, using convex

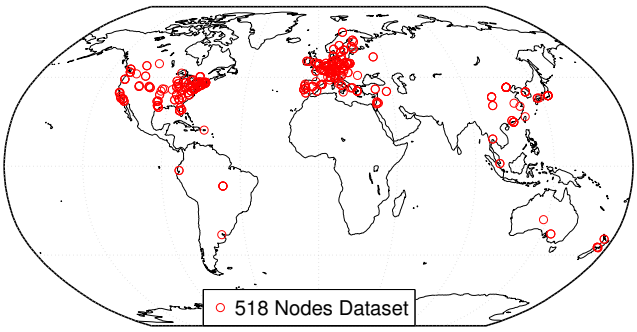


Figure 2: The locations of 518 PlanetLab nodes.

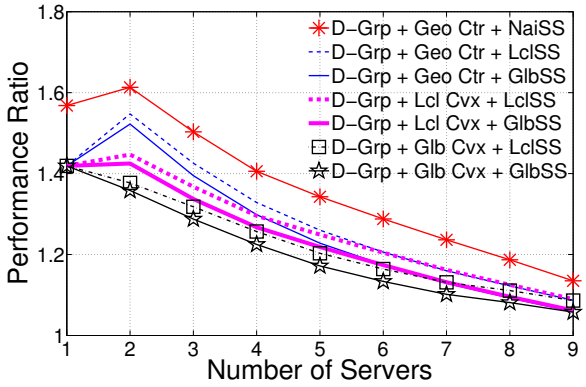


Figure 3: The mean end-to-end delay (normalized by the full-mesh mean delay) of different methods for 12 clients.

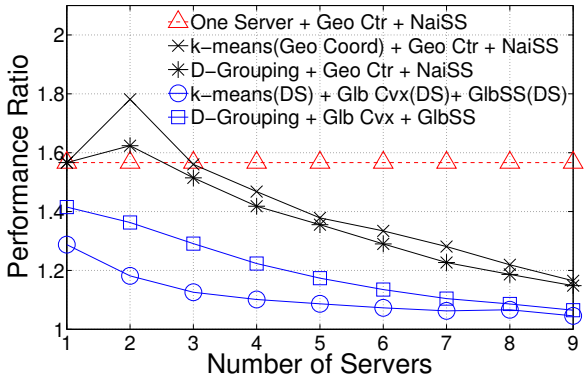


Figure 4: The mean end-to-end delay (normalized by the full-mesh mean delay) for one server, the delay-space method via Vivaldi and our methods for 12 clients. DS: delay-space method.

optimization to tune server locations brings salient benefits. Comparing the all the methods using D-Grouping and Global Server Search but different ideal server location optimization methods, we notice that Global Convex Optimization for server locations has the best performance, followed by Local Convex Optimization, which are much better than Geo-Center. *Secondly*, the delay can be effectively reduced by introducing 3 candidate servers during the mapping phase, using Local Server Search and Global Server Search. For example, if we check all the methods using D-Grouping and Geo-Center, we observe that Global Server Search is a bit better than Local Server Search, and much better than Naive Server Search. Overall, the method of using D-Grouping + Global Convex Optimization + Global Server Search is the best method that incurs the lowest delay, although it is more complex than its local counterpart.

We compare the performance of a one-server (geo-center) solution, delay-space solution via Vivaldi, and three of our methods for 12 clients in Fig. 4. The delay-space approach uses Vivaldi to embed clients into a 5-D delay space (network coordinate system), uses k-*means* to divide clients into groups, and then uses Global Convex Optimization to find the set of ideal server locations in the delay space directly. After that, real servers are selected by using Global Server Search also performed in the delay space directly. The first of our methods is a naive benchmark method which uses k-means to divide clients into groups based on their geographic locations, and our second method uses D-grouping to partition clients. Both select a server closest to the geographic center of each group as the server for that group (Geo-Center + Naive Server Search). Our third method in this figure uses D-Grouping, Global Convex Optimization for server locations, and Global Server Search.

Firstly, we notice that all multi-server methods are better than the one-server (geo-center) solution when the number of servers is greater than 2. However, when only 2 servers are allowed, the performance of some methods is worse than one server. The reason is that the benefits of using multi-servers cannot offset all the inaccuracies introduced, including mapping errors in the server search, and the mismatch between geo-distances and network distances, etc. *Secondly*, we observe that D-Grouping, as a delay-based method, is always better than k-means performed on geographic coordinates. *Furthermore*, we find that D-Grouping + Global Convex Optimization + Global Server Search, *although boldly utilizing geo-information in its last 2 steps*, has comparable performance to the delay-space-based method via Vivaldi, yet without having to collect the pings of all available servers or having to embed nodes into a delay space.

Furthermore, Fig. 5 shows the delay performance as the number of servers shared per client changes for our best method: D-Grouping + Global Convex Optimization + Global Server search. Complying to the intuition, the delay decreases as the number of servers shared per client increases for 4—20 clients. However, it is interesting to note that when the number of servers per client is fixed, the more servers used, the lower the delay. Furthermore, the marginal benefit of increasing the number of servers per client actually decreases, leading to a convex-shaped trend.

Finally, Table 1 shows the time consumption of some selected algorithms for 12 clients on a 2.3 GHz quad-core (i7-3615QM) processor. We observe that Global Convex Optimization consumes the most of the time in the scheme. Note that it only takes about 3 seconds to compute the best 6 servers for 12 clients, which is a large participation already for today's multiparty conferences. Even adding the online ping collection time (performed in D-Grouping and

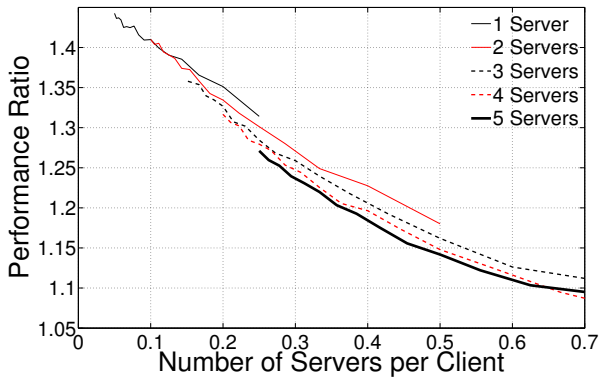


Figure 5: The mean end-to-end delay (normalized by the full-mesh mean delay) vs. the number of servers shared per client with our best method for 4—20 clients and 1—5 servers.

# Servers	1	2	3	4	5	6
D-Grouping	0	2.3	2.7	3.3	4.0	4.4
Glb Convex Opt	1.2	189	564	1,092	1,850	2,689
Glb Server Search	0.8	68.8	87.9	127	225	525
Total (ms)	2.0	260	654	1,223	2,079	3,219

Table 1: Time consumption (ms) breakdown of different algorithms for 12 clients.

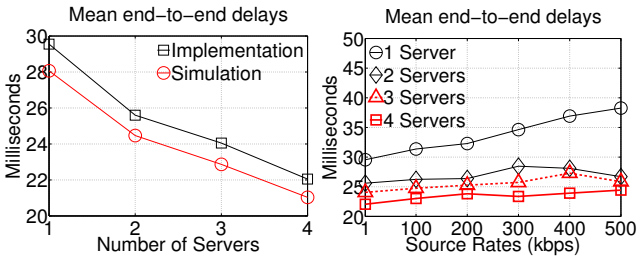
the server search (mapping) phase in parallel), the total session preparation time will not exceed several seconds.

5. PROTOTYPE IMPLEMENTATION

To verify the real-world performance of the proposed methods, we used the *Apache Thrift* framework and the Boost library to develop an asynchronous multi-threaded packet communication module in 2,000 lines of C++ code. And Thrift generated about 5,000 lines of C++ code. We deployed this conferencing system on PlanetLab nodes. In our experiments, the frequency at which each client sends packets to its corresponding server is set to 300 packets/second, where the source rate is controlled by tuning the packet size: $\mathsf{sourcerate} = \mathsf{packetsize} \times \mathsf{frequency}$. The payload in each packet contains random characters. We aim to measure the real packet-level end-to-end delays achieved under different methods at different source rates.

5.1 Measuring End-to-End Packet Delays

Since it is hard to synchronize the clocks on different computers, the delay between clients (on the order of ms) cannot be measured by simply recording the sending time on the sender and the receiving time on the receiver. We propose an indirect method to measure the end-to-end delay of each packet. Suppose client A is sending packets to another client B via some servers. At the very moment before A sends out a packet, it starts a timer. When the packet eventually reaches B, B will send a ping packet *directly* to the sender A. When A receives the ping packet, it stops its timer and record the time span T_{circle}, which is the end-to-end delay of the packet from A to B plus the one-way ping time from B to A. When B gets the reply of the ping from A, it records



(a) Implementation (1 kbps) v.s. simulation (b) Implementations at different source rates

Figure 6: The mean end-to-end delays in implementation at various source rates for 6 clients.

the round trip time RTT_{AB} . Therefore, the end-to-end delay from A to B can be evaluated by $T_{\mathsf{circle}} - \mathrm{RTT}_{AB}/2$. In our implementation, each client measures the end-to-end delay to all other clients once every 300 packets.

5.2 Real-World Experiments

We randomly select 6 PlanetLab nodes as clients and use our best method, namely D-grouping (ping-based) + Global Convex Optimization (geo-based) + Global Server Search (ping-based), to output the best server nodes, as the server number varies from 1 to 4. Then, we deploy our distributed communication systems on these 6 client nodes as well as on the selected server nodes, and measure real end-to-end delays between clients.

Fig. 6(a) shows the performance comparison between simulation and implementation. To eliminate the influence of source rates on latency , here we deliberately set the source rate to be 1 kbps. Note that as the number of servers increases, the real delay (ms) in the implementation drops at a similar pace to that of the simulation (which estimates delays by summing up RTTs). The real delay is only slightly worse than the simulated result due to the existence of queuing delays and processing (CPU) delays.

Fig. 6(b) illustrates the change of the mean end-to-end delays as the source rates of clients increase. When the number of servers is 2, 3 and 4, the mean end-to-end delays only increases slightly as the source sending rate increases from 1 kbps all the way to 500 kbps at each source (which can support sufficiently high video quality). However, in the one-server solution, as the source rate increases, the mean end-to-end delays has a dramatic increase, which surges from 443 ms to 574 ms as the source rate changes from 1 kbps to 500 kbps. The reason is that the server uploading burden increases with fewer servers. For example, in the toy example of Fig. 1(c) and Fig. 1(d), suppose each of the 4 clients is about to send a packet to all other clients. In Fig.1(c), the server has to collect the 4 packets and send them out for 12 times. However in Fig.1(d), every server collects 4 packets but only sends them out for 8 times. As a result, using multi-servers relieves both the network and CPU burdens at each server, leading to dramatically less increase in processing and queuing delays as throughput grows.

6. RELATED WORK

Video conferencing has been extensively studied in the context of P2P networks [1, 5]. These works seek to optimize the streaming rates of all the peers subject to network

bandwidth constraints in a utility maximization framework. Recent work uses the cloud to enhance the performance of video conference sessions. Airlift [3] uses inter-datacenter networks to relay traffic and process data streams in video conferencing. It maximizes the total throughput in multiple conference sessions by choosing the optimal way to deliver and relay packets in the cloud, subject to end-to-end delay constraints. In contrast, our work focuses on minimizing the end-to-end delays in an individual conference session.

Our work is related to [6], which also uses multi-servers, called a Virtual Mixer, to reduce delay in video conferencing. In particular, it tries to minimize either the average or the maximum end-to-end delay using a heuristic based on Steiner tree optimization performed on a graph of servers and clients. However, this heuristic is not scalable to a graph of thousands of servers. In contrast, our simple algorithms optimize the mean end-to-end delay in a geometric problem instead of on a graph, and easily scale to any number of servers.

Navigator [8] is used to estimate the network proximity/latency. It can find the actual closest node with 90% confidence. However, considering the cost of running it on the cloud, Navigator is not the best choice in our system.

Network coordinate (NC) system is an efficient mechanism for Internet latency estimation. Vivaldi [2] is a representative distributed NC system, and is deployed in many well-known Internet systems, e.g., Azureus BitTorrent [4]. Measurements [2, 7] show that there is some correlation between the pairwise delay and pairwise geo-distance of two hosts, which supports the feasibility of our server location optimization without knowing information about delays to all the servers. In experiments, our simple method performs well even without resorting to the delay-space embedding.

7. CONCLUDING REMARKS

This paper studies the placement of multi-servers and topology control in multi-party video conferencing applications. We propose D-Grouping to group clients given their pairwise ping statistics, assign each client group a server, and use optimization to fine-tune the server locations with the objective of reducing end-to-end delays. We evaluate our methods based on ping traces collected from 518 PlanetLab nodes and show that our proposed methods have comparable performance to full network coordinate embedding in a delay space, yet with much lower overhead. Experiments based on our prototype system further suggests that not only do multi-servers reduce delay in multi-party video conferencing over the industrial state of the art that uses a single server, but they can also support higher throughput in practice under reasonable end-to-end packet delays.

8. ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their valuable opinions to improve the final version of this paper, especially for pointing out a few pieces of missing related work. This research was partly supported by the Discovery Grant from Natural Sciences and Engineering Research Council (NSERC) of Canada, and partly supported by a grant from National Natural Science Foundation of China under grant No. 61370232.

9. REFERENCES

[1] X. Chen, M. Chen, B. Li, Y. Zhao, Y. Wu, and J. Li. Celerity: A low- delay multi-party conferencing solution. In *Proc. of ACM Multimedia*, 2011.

[2] F. Dabek, R. Cox, F. Kaashoek, and R. Morris. Vivaldi: A decentralized network coordinate system. In *Proc. of ACM SIGCOMM*, 2004.

[3] Y. Feng, B. Li, and B. Li. Airlift: Video conferencing as a cloud service using inter-datacenter networks. In *Proc. of IEEE ICNP*, 2012.

[4] J. Ledlie, P. Gardner, and M. Seltzer. Network coordinates in the wild. In *Proc. of NSDI*, 2007.

[5] C. Liang, M. Zhao, and Y. Liu. Optimal bandwidth sharing in multiswarm multiparty p2p video-conferencing systems. *IEEE/ACM Trans. Networking*, 19(6):1704–1716, 2011.

[6] J. Liao, C. Yuan, W. Zhu, and P. A. Chou. Virtual mixer: Real-time audio mixing across clients and cloud for multi-party conferencing. In *Proc. of IEEE ICASSP*, 2012.

[7] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *Proc. of ACM SIGCOMM*, 2001.

[8] P. Sharma, Z. Xu, S. Banerjee, and S.-J. Lee. Estimating network proximity and latency. *ACM SIGCOMM Computer Communication Review*, 36(3):39–50, 2006.