DIGIPARTY – A DECENTRALIZED MULTI-PARTY VIDEO CONFERENCING SYSTEM

Ling Chen, Chong Luo, Jiang Li, and Shipeng Li

Microsoft Research Asia

ABSTRACT

Increased speeds of PCs and networks have made media communication possible on the Internet. However, nearly ten years after the first release of Microsoft NetMeeting, Internet video telephony is still limited to the point-to-point communication mode. Today, people have a need for an easy-to-use multi-party video conferencing tool that will connect families and friends around the world over the Internet. In this paper, DigiParty, a fully distributed multi-party video conferencing system, is presented. DigiParty employs full mesh conferencing architecture and adopts a loosely coupled conferencing mode. A novel conference control protocol is designed with the system. DigiParty can be integrated with any existing instant messaging services and is applicable to all types of Internet connections.

1. INTRODUCTION

When Microsoft NetMeeting [1] was first released in 1996, it was the first commercial real-time communication client that enables more than two users to share data over the Internet. A few months later, the video communication service was integrated into NetMeeting. Although people then expected a multi-party video conferencing system that enables families and friends around the world to keep in touch over the Internet, almost ten years have passed and Internet video telephony is still limited to the antiquated point-to-point mode. Multi-party video conferencing systems are only available on LAN (e.g. MERMAID [2]), ATM (e.g. GCSVA [3]) or MBone (e.g. VIC [4]). In recent years, with the evolution of broadband technology, there have emerged several multi-party video conferencing systems running on WAN ([5], [6]). However, none is ready to be used by the general public for everyday communications.

iVisit [5] is a server-based communication tool. Central servers are used to provide membership registration and verification. A user can set up an A/V session with any buddy on his friend list. Other than Netmeeting, iVisit allows a user to make multiple connections as well as view multiple videos at the same time. However, if we want to hold a multi-party conference, connections have to be manually made one by one. On the other hand, iVisit maintains an Internet video community, which is partitioned into different categories and separate rooms. Each room is a rendezvous point, where each user is aware of any other presences in the same room. In this mode, although instant messages can be made to be seen publicly, video communications are based on independent one-to-one connections only. Conference control mechanisms are not supported in both modes of iVisit. Strictly speaking, iVisit is not a conferencing application; rather, it is just a multiple one-to-one communications tool.

WebEx [6] is another famous system that provides online meeting services for global business. A powerful communications tool, it supports a full range of online meeting services, such as application sharing, white board, and video conferencing. However, a general purpose system can never excel in every specific feature. The video conferencing service, therefore, is just an assistant tool in WebEx, and is mutually exclusive with participants label and polling label. WebEx leverages the MediaTone technology to deliver media data. The idea is to build up a number of switching centers worldwide, which are responsible for routing communications among end users. This strategy is efficient for a large number of simultaneously video conferences. However, they are technically over-engineered for small-scale personal communications. Even in the case where two members are in the same domain, data is still transmitted from switching centers, which incurs an intolerable delay for video communications.

It has been noticed that most video conferencing approaches so far employ central servers either for conference management or for data distribution. Obviously, such an approach may create performance bottlenecks for a large number of participants. Furthermore, the cost of maintaining central servers makes the service too expensive to be used by individuals. As an alternative, distributed architectures can resolve the problems mentioned above. There are two kinds of distributed architectures which are slightly different from each other. One is peer-to-peer conferencing, where each end system has a direct connection with only a subset of its peers and talks with other members through a sequence of intermediate hops.

The other distributed architecture is full mesh conferencing, where each conference member has direct communication channel with each other. While peer-to-peer conferencing is suitable for large scale video conferencing systems, full mesh conferencing is more suitable for small scale systems because of its simplicity in control. In this paper, DigiParty, a multi-party video conferencing system is presented. It is a fully distributed system built on full mesh conferencing architecture. A specific conference control protocol is designed for it. The rest of this paper is organized as follows. Section 2 describes the system architecture of DigiParty. In Section 3, a new conference control protocol is illustrated. Section 4 presents extensive experimental results on protocol verification. We conclude and indicate future research directions in Section 5.

2. DIGIPARTY – A MULTI-PARTY VIDEO CONFERENCING SYSTEM

DigiParty is not only a protocol but also a real system that is ready to be used for everyday communications. It supports all types of Internet connections, including LAN, broadband, and even dial-up. It can be integrated with all kinds of instant messaging services, and a version for MSN messenger has been developed. Figure 1 depicts the system architecture of DigiParty.

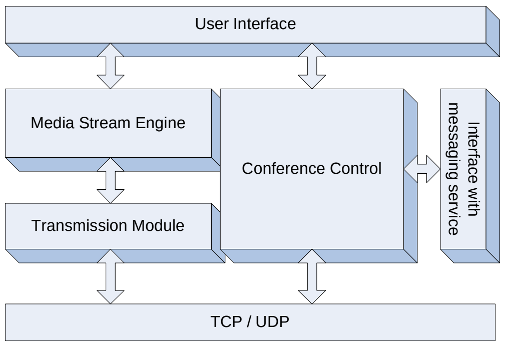

Figure 1: The system architecture of DigiParty

DigiParty is composed of five modules. The core modules are the media stream engine and the full mesh conferencing protocol. The media stream engine is responsible for audio/video capture and playback. The conferencing control defines a set of conventions governing the structure and behavior of communication messages. The details of the protocol will be introduced in the next section.

The full mesh conferencing structure is first introduced in [7], where Lennox et al. also point out that the full mesh conferencing architecture is not suitable for bandwidth-limited end systems, such as wireless devices and users with 56 kbps modems. To break this limitation, in our system, we entirely separate the transmission module from the media stream engine and define a whole set of APIs that are open for both Unicast and application-level multicast (ALM). When there are multiple data receivers, multicast allows data replication to be performed outside of the data source. Application-level multicast is different from traditional IP multicast in that data replication is conducted at end systems instead of multicast-enabled routers. With a proper ALM algorithm, we are able to alleviate the scalability problem of full mesh conferencing architecture.

The interface with messaging services is one of the most important features of DigiParty. It is designed out of usability considerations: People today are overburdened by dozens of accounts and passwords in multifarious Internet tools or communities. This can even cause hesitation in signing up for new services. Most Internet users are just using instant messaging services (MSN, Yahoo, etc.) to communicate with their families, friends and co-workers. Based on this observation, we arranged a few pins attached to our main modules so that DigiParty can be plugged into any existing instant messaging services, as long as an open programming interface is provided. Presently, the version that works with MSN Messenger has been implemented. DigiParty is able to extract a local user's MSN buddy list, from which the conference initiator can choose the desired attendees. Then, invitations are sent via MSN instant messages, which also include the inviter's IP address and a unique conference ID. Upon receiving such an invitation, the user can join the conference by simply clicking on the link provided in the instant message.

3. CONFERENCE CONTROL PROTOCOL

Much research has been done on conference control protocols, such as the conference control channel protocol (CCCP) [8], the simple conference control protocol (SCCP) [9], and a protocol for reliable decentralized conferencing [7]. Among all these protocols, [7] is architecturally closest to ours. However, our protocol is designed along with the application scenarios and it should provide the quality of service (QoS) needed by a multi-party video conferencing system. As we all know, video conferencing is highly bandwidth demanding and has very stringent requirements on transmission delay. No matter which media transmission structure we use, Unicast or application-level multicast, more members in a conference leads to a lower level of QoS. Therefore, in our protocol, we provide a mechanism to control the number of conference members, so that we can provide better service to those members who joined the meeting earlier than others. While we use the limit of 5 in our implemented system, this number is flexible and can be adjusted to the user's satisfaction.

In the next subsections, we will first give an overview of the protocol, after which the design challenges and our solutions will be stated.

3.1. Full mesh control protocol

Our protocol is designed based on the full mesh architecture, where conference members are united by a fully connected communication mesh. And all the members are equivalent in terms of position in topology or rights in the conference. Different from [7], our protocol is so concise that it uses only four communication messages:

- ♦ ***JOIN***: This message is sent from a new user (e.g. *N*) to an existing conference member (e.g. *M*). It contains the member information of the new user, such as the display name in MSN messenger.
- ♦ ***ACCEPT***: This message is in reply to the *JOIN* message, if the conference member *M* wants to accept the newcomer *N*. It consists of *M*'s member information, as well as the member list in *M*'s view.
- ♦ ***REJECT***: This message is also in reply to the *JOIN* message, if for any possible reason, member *M* rejects *N*'s join request.
- ♦ ***LEAVE***: This message is sent from a leaving member to all the other conferencing members to politely inform them of his leaving.

A typical scenario is when a new user *N* is invited to an existing conference. *N* is required to build communication channels with all the other conference members in order to keep the full mesh complete. Figure 2 illustrates the process of making connections between *N* and *M* (an existing member). Supposing there are three attendees (*A*, *B* and *C*) in this conference before *N* joins, this process will be executed three times.

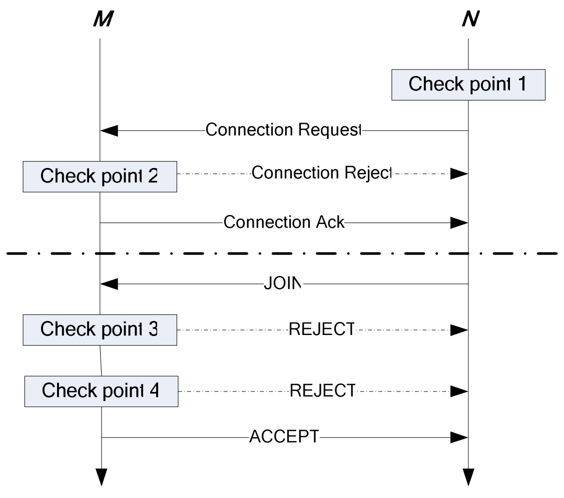

Figure 2: The process of building communication channel

In this figure, the two vertical lines are the time line. *M* is an existing conference member and *N* is a new joiner. The actions above the bold dotted line make a TCP connection, while the actions below build a communication dialog. When *N* accepts an invitation, it sends a Connection Request to its inviter. If the request is acknowledged, *N* will send a *JOIN* message to *M*. *A* respond with an *ACCEPT* message that lists the IP address of all the conference members (i.e. *A*, *B* and *C*) in the conference. *N* receives the *ACCEPT* message and connects with members who are not in its connection list yet. In this way, a full connection mesh of four members is built.

Please note that there are four check points listed in this figure. They are designed to deal with concurrency problems. If any of these check points fails, the process will terminate and the new member *N* will not be able to join the conference.

3.2 Concurrency problems

The most challenging task in a loosely coupled full mesh protocol is to keep the connection mesh complete, since it allows any member in a conference to introduce new users at any time. In our protocol, we ensure that each member has communication channels with all the others, and at the same time keep the number of conference members below a given limit. This is difficult for a decentralized system, especially when there are concurrent operations by different users or even the same user. Here, concurrency does not mean exactly coincident. Instead, we define:

Concurrent actions: if action *B* occurs when the conference is in an unstable state which is caused by action *A*. Then we say that action *A* and *B* are concurrent actions.

3.2.1 Concurrent joining:

If two members *E* and *F* join a conference concurrently, chances are: 1) *E* and *F* have double connections with each other; 2) Number of conference members exceeds the limit. The first problem happens in such a scenario: member *A* and *B* are in a conference, and *A* invites *E* while *B* invites *F*. In the *ACCEPT* message from *A* to *E* and *B* to *F*, *E* and *F* are simultaneously informed of the existence of each other. So both of them send connection request to the other, and the request is accepted on both sides. Although double connections do not harm the full mesh structure, it involves additional resource usage and thus is undesirable. We solve this problem by introducing a pending list. When a member (e.g. *E*) sends a Connection Request (CR in short) to another (e.g. *F*), it will keep *F*'s unique identification in its pending list. It moves *F* to the member list only after the CR is accepted. If *E* receives a CR from *F* while *F* is in its pending list, it will compare *F*'s identification with its own and make accept or reject decision according to the result (**Check point 2**). In our system, we use the dotted IP address as member identification and accept a CR from a member in pending list only if its ID is larger than the local ID.

The second problem happens in a quite similar scenario. Suppose there are two other members *C* and *D* in this conference and the conference number limit is five. If both *E* and *F* join this conference, the limit will be violated. Thus, we require each member to check the length of its member list before accepting a new user (**Check point 4**). Moreover, we restrict the number of new users to be invited in conformity to the regulation. In the given example, *A* is not allowed to invite two guests since there are already four attendants in the conference.

3.2.2 Concurrent joining and leaving

Concurrent joining and leaving may refer to different users or the same user. In the former case, let us consider the following situation: originally *A*, *B* and *C* are in a conference. Then, *A* wants to leave and is sending *LEAVE* messages to the others while *D* is invited to the conference by member *B*. It is possible that *A* receives *D*'s connection request while it is trying to exit the application. If *A* just follows the process as we described above, it will re-join the conference and is not able to leave. This problem can be resolved by adding a presence flag. If *A* decides to leave, it will turn the flag to FALSE so that it will check this flag before accepting any connection request (**Check point 3**).

Another situation that may cause problems is when a user leaves a conference shortly before it decides to re-join. In this case, if the *JOIN* message outraces the *LEAVE* message, other members will ignore the first arrived *JOIN* message and close the connection when the *LEAVE* message arrives. This is an undesirable result. Since DigiParty ensures that there is only one application instance running on a single machine (**Check point 1**), we can solve this problem by the following strategy: if a member receives the *JOIN* message from a member that is already on its member list, it will close the previous connection and accept the current request. The reason is very straightforward.

3.3 Security issues

Without security protection, DigiParty is open to intruders on the Internet and is vulnerable to malicious attacks. In this section, we offer a simple mechanism to avoid such attacks. Each conference has a unique 128-bit conference ID, which is similar to the GUID (Global Unique Identifier). This ID is generated by the conference initiator before he is able to invite any buddies to attend the conference. In all the invitations regarding the same conference, the conference ID will be carried in the instant message. Although the invitation message may be overheard during its transmission over the Internet, we believe that this level of protection is adequate for a conferencing system that mainly serves for personal communications. Since computer security in the real world is not only the matter of locks, but also related to the comparison of value and costs.

4. EXPERIMENTS ON PROTOCOL VERIFICATION

The verification of a conferencing protocol, especially a multi-party conferencing protocol, is difficult. The reason is that a protocol's behavior strongly depends on the order in which events occur, and the number of possible orders is, in fact, exponential in the size of the group and the number of actions. Thus, it is not always possible to use a custom program to explore every possibility, as the potential conference size is unlimited.

Our verification work is done by validating the conference control protocol in all possible concurrent scenarios. As we have mentioned in the previous section, the concurrency of joining and leaving events tends to create instability in a conference. While concurrent joining can certainly happen on distinct ends, concurrent joining and leaving may happen on the same machine, as well as on different machines. Therefore, we define four types of events: JOIN, LEAVE, BRIEF JOIN (join and then leave immediately) and BRIEF LEAVE (leave and then re-join immediately). Thus, all the concurrency situations are the combinations of these four events, which are summarized in Table 1.

Table 1: Concurrency tests

Test	Initial State	**Action**	Final State
1	(A,B,C,D)	**J(E, F)**	(A,B,C,D,E) (A,B,C,D,F) (A,B,C,D)
2	(A,B,C,D)	**L(C,D)**	(A,B)
3	(A,B,C,D)	**BJ (E)**	(A,B,C,D)
4	(A,B,C,D)	**BL(D)**	(A,B,C)
5	(A,B,C)	**J(D), L(C)**	(A,B,D)
6	(A,B,C)	**J(D), BJ(E)**	(A,B,C,D)
7	(A,B,C)	**J(D), BL(C)**	(A,B,C,D)
8	(A,B,C,D)	**L(D), BJ(E)**	(A,B,C)
9	(A,B,C,D)	**L(D), BL(C)**	(A,B,C)
10	(A,B,C,D)	**BJ(E), BL(D)**	(A,B,C,D)
11	(A,B,C)	**J(D), L(C), BJ(E)**	(A,B,D)
12	(A,B,C,D)	**J(E), L(D), BL(C)**	(A,B,C,E)
13	(A,B,C)	**J(D), BJ(E), L(C)**	(A,B,D)
14	(A,B,C,D)	**L(D), BJ(E), BL(C)**	(A,B,C)
15	(A,B,C)	**J(D), L(C), BJ(E), BL(B)**	(A,B,D)

In this table, each row represents a test run. In the initial state, some members are already in the conference. Then, actions listed in the 3rd column are taken place concurrently. Here, J, L, BJ and BL represent JOIN, LEAVE, BRIEF JOIN and BRIEF LEAVE respectively. A valid protocol should lead the conference to the final state described in the last column. All these tests have a definite final state except for the first one. As mentioned earlier, we have a mechanism to control the number of conference members and the current limit is 5. Thus, in the first test run, only one of the two joiners is allowed to attend the conference. There is an extreme case when the two joiners are invited by different members and each inviter tries to "protect" his own invitee, our protocol rejects both joining attempts and allows the inviters to negotiate before they send another invitation.

We did experiments for all the 15 situations. After each run, we carefully check the connection status at every live site. With no surprise, our protocol passed all the 15 tests and every conference member stabilizes with exactly a single active connection with every other member.

5. CONCLUSION

In this paper, we presented a multi-party video conferencing system named DigiParty. It is a powerful extension to existing messenger services and can be used by most Internet users including dial-up users. The accompanying conferencing protocol is elaborately designed to handle all the possible concurrent situations. The chief advantages of DigiParty arises from its reliability, flexibility and low cost. We believe that DigiParty will bring Internet video telephony to a new level of quality and will lead to a new trend in everyday communications.

Future works may be conducted in the following directions: 1) Increasing the security level so that DigiParty can be used in the field of business; 2) Transplant DigiParty to mobile devices and enable mobile conferencing.

6. ACKNOWLEDGEMENT

The authors would like to acknowledge Keman Yu for helpful discussions and Steve Lin for proofreading the paper.

7. REFERENCES

[1] NetMeeting, http://www.microsoft.com/netmeeting/

[2] J. S. Park, S. H. Lee, S. C. Kim, J. Y. Lee, S. B. Lee, "A conferencing system for real-time, multiparty, multimedia services", *IEEE Transactions on Consumer Electronics*, pp. 857-865, Vol. 44, No. 3, 1998.

[3] I. Beier, H. Koenig, "GCSVA - a multiparty videoconferencing system with distributed group and QoS management", *In Proc of 7th International Conference on Computer Communications and Networks*, pp. 594-598, 1998.

[4] Networked Multimedia Research Group at University College London, Video Conferencing Tool, http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/

[5] iVisit, http://www.ivisit.info/

[6] WebEx, http://www.webex.com/

[7] J. Lennox, H. Schulzrinne, "A protocol for reliable decentralized conferencing", *In Proc of 13th international workshop on network and operating systems support for digital audio and video*, pp. 72-81, 2003.

[8] M. Handley, I. Wakeman, J. Crowcroft, "The conference control channel protocol (CCCP): a scalable base for building conference control applications", *In Proc of ACM SIGCOMM*, pp. 275-287, 1995.

[9] C. Bormann, J. Ott, C. Reichert, "SCCP: simple conference control protocol", Internet Draft. Draft-ietf-mmusic-sccp-00.txt, Work in Progress, June 1996.