Lightweight Peer-to-Peer Secure Multi-Party VoIP Protocol

JOSÉ-VICENTE AGUIRRE[1], RAFAEL ÁLVAREZ[2], LEANDRO TORTOSA[3]
and ANTONIO ZAMORA[4]
Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante
Campus de Sant Vicent del Raspeig, Ap. Correos 99, E-03080, Alicante
SPAIN
jaguirre@dccia.ua.es[1] ralvarez@dccia.ua.es[2] tortosa@dccia.ua.es[3] zamora@dccia.ua.es[4]

This work was partially supported by the Spanish grants GV06/018 and MTM2005-05759

Abstract: - Multi-party voice-over-IP (MVoIP) services provide economical and convenient group communication mechanisms for many emerging applications such as distance collaboration systems, on-line meetings and Internet gaming. In this paper, we present a light peer-to-peer (P2P) protocol to provide MVoIP services on small platforms like mobile phones and PDAs. Unlike other proposals, our solution is fully distributed and self-organizing without requiring specialized servers or IP multicast support.

Key-Words: - Multiparty VoIP, MVoIP, VoIP, P2P, Security, Pocket PC, Smart Phone.

1 Introduction

Voice over Internet Protocol [1] (also called VoIP, IP Telephony, Internet telephony, and Digital Phone) is the routing of voice conversations over the Internet or any other IP-based network.

Secure VoIP includes a Session Initiation Protocol [6] with session key transport and providers like Skype [7] have already been around for a little while and are growing steadily.

Multi-party voice-over-IP (MVoIP) are VoIP services that can include three or more participants. Traditional conferencing systems often employ IP multi-cast (e.g., [3]) or overlay multicast ([2], [4]).

Although the multicast approach is well suited for broadcast applications that usually involve one active speaker, it becomes inefficient for interactive and spontaneous applications (e.g., distance collaboration systems, on-line meetings and Internet gaming) that often include many simultaneous speakers.

In this paper, we propose a new P2P audio stream processing system to balance computational and network resources load around all machines involved in MVoIP communication.

Compared to the current state of the art [5], this approach provides three novelties:

- First, the protocol performs a fair load distribution of the data mixing and transmission operations.
- Second, the protocol is fully distributed and self-organizing.
- Finally, the protocol guarantees that the audio mixing phase produces the audio distribution implicitly.

2 Notation

We use the following notation in this paper:

- N is the total number of machines.
- n is the current machine.
- I is the total number of iterations of the algorithm.
- i is the current iteration of the algorithm.
- V_n is the current voice packet of node n.
- V_a is the fully mixed voice packet ready for playback.
- $\underset{y=0}{\overset{x}{Mix}}(V_y)$ is a packet mixing function that mixes from y=0 to x.
- *Parallel {{Job1} {Job2}}* denotes that jobs 1 and 2 are carried out in parallel.

Unless otherwise specified, all variables are non negative integer numbers and operations like logarithms provide float results.

3 Protocol specification

In the following, we describe the protocol specification and requirements.

3.1 Requirements

The protocol aims to be a valid proposal for environments in which there are N machines that want to establish a common audio channel, so that any machine can listen to what the others are saying at any moment, even if all of them were to communicate simultaneously.

This hypothesis is done considering the most restrictive conditions possible regarding the machines and the transmission channel, which are: first, that all machines have limited computational resources, so that they are not capable of being the central server or super-node; and second, that the transmission channel can only support one simultaneous transmission and reception. These restrictions are easily found in the case of communicating multiple mobile phones or PDAs.

The study of the impact that relaxing some of these restrictions would have on the protocol design is out of the scope of this paper.

3.2 Definition

The protocol is defined as a packet mixing and distribution algorithm in a network of *N* machines.

The general algorithm shows adequate packet distribution behaviour in the case that $N = 2^I$, but a more detailed study is necessary when this is not true.

There is a subset of these cases for which the algorithm can be adapted without any performance impact; for the rest of these cases that do impact performance, we present several possible alternatives giving as a result an adapted version of the algorithm.

3.2.1 General algorithm

In the case of having *N* machines connected in a virtual ring, with sequential numbering, so that each machine has a fixed number from 0 to *N*-1, we can establish the emitting and receiving nodes with

$$N_e(n,i) = n + 2^{i-1} \bmod N \qquad (1)$$

and

$$N_r(n,i) = n - 2^{i-1} \bmod N, \qquad (2)$$

being $N_e(n, i)$ (see (1)) the node to which *n* must send P_e (see (9)) during iteration *i*; and $N_r(n, i)$ (see (2)) the node from which n must receive P_r (see (8)) during iteration *i*.

With the previous specifications, we can define an algorithm (see Fig. 1) whose mixing and distribution characteristics are defined in equations (3) to (7).

$$P_e(n,i) = \underset{y=0}{\overset{2^{i-1}-1}{Mix}}\left(V_{(n-y) \bmod N}\right) \qquad (3)$$

$$P_r(n,i) = \underset{y=0}{\overset{2^{i-1}-1}{Mix}}\left(V_{((n-2^{i-1})-y) \bmod N}\right) \qquad (4)$$

$$P(n,i) = \underset{y=0}{\overset{2^{i}-1}{Mix}}\left(V_{(n-y) \bmod N}\right) \qquad (5)$$

$$V_a(n,i) = \underset{y=1}{\overset{2^{i}-1}{Mix}}\left(V_{(n-y) \bmod N}\right) \qquad (6)$$

$$DV_a(n) = \underset{y=1}{\overset{N-1}{Mix}}\left(V_{(n-y) \bmod N}\right) \qquad (7)$$

In this way, we can define the following:

- $P_e(n,i)$ (see (3)) corresponds to the composition of the packet that node *n* will have to send during iteration *i*.
- $P_r(n,i)$ (see (4)) corresponds to the composition of the packet that node *n* will have to receive during iteration *i*.
- $P(n,i)$ (see (5)) is the final packet that node *n* will have composed after the reception of the last packet during iteration *i*.
- $V_a(n,i)$ (see (6)) is the accumulated composition of the voice packet for playback at node *n* during iteration *i*. V_a differentiates from *P* in that it does not include V_n.
- $DV_a(n)$ (see (7)) is the desired accumulated voice packet composition for playback at node *n* during iteration *i*.

```
Function TransmitVoice (VoicePacket myVoice, int
numNodes, int myPosition)
{
  N= numNodes;
  n= myPosition;
  AllPacketReceived.add ( myVoice );
  For (i=1; i <= log2(N); i++)  {
    NodeDestination =  n + 2^(i-1);
    NodeOrigin = n - 2^(i-1);

    Parallel    {{ PacketReceive = receive ( NodeOrigin );
      AllPacketReceived.add (   Mix (PacketReceive,
AllPacketReceived [i-1] ) );  }
     { PacketSend =  AllPacketReceived [i-1];
      Send(NodeDestination, PacketSend);   } }
  }
}
```

Fig. 1 *General algorithm*

In a recursive way, closer to the real behaviour of the algorithm, the previous functions can be defined as shown in equations (8) to (11).

$$P_r(n,i) = P_e(N_r(n,i),i) \qquad (8)$$

$$P_e(n,i) = P(n,i-1) \qquad (9)$$

$$P(n,i)=\begin{cases} Mix\left(P(n,i-1),P_r(n,i)\right) & if \quad i>0 \\ V_n & if \quad i=0 \end{cases} \qquad (10)$$

$$V_a(n,i)=\begin{cases} Mix\left(V_a(n,i-1),P_r(n,i)\right) & if \quad i>0 \\ \{\} & if \quad i=0 \end{cases} \qquad (11)$$

Employing equation (6) we can obtain the table shown in Fig. 3 that represents the audio packets mixed for playback at machine *n* with a total of *N* machines.

This table represents the values of $V_a(n,i)$ for node $n = N$-1 because it is the most clear case, having V_k values decreasing from $k = N$-2 to $k = 0$.

Observing this table (see Fig. 3), we can extract three different cases as a function of the correspondence of the generated V_a with the DV_a (desired V_a see equation (7)).

The first case is when $N = 2^I$.

The second case is when $N <> 2^I$ *and* $N = 2^{I-1} + 2^x$, where $x < I$.

The third case is when $N <> 2^I$ and $N <> 2^{I-1} + 2^x$ where $x< I$. In the following, we provide a detailed study of each one of these cases.

3.2.2 Case $N = 2^I$

These are the base cases of the algorithm and do not require any modification to the general algorithm to be treated.

3.2.3 Case $N <> 2^I$ and $N = 2^{I-1} + 2^x$ where $x < I$

In this case, with a slight modification of the general algorithm we can achieve the same performance than in the base case. To do so, it is necessary to modify P_e (equation (13))

$$x = \log_2\left(N - 2^{I-1}\right) \qquad (12)$$

$$P_e(n,i)=\begin{cases} P(n,i-1) & if \quad i<>I \\ P(n,x) & if \quad i=I \end{cases} \qquad (13)$$

3.2.4 Case $N <> 2^I$ and $N <> 2^{I-1} + 2^x$ where $x < I$

In this case, we cannot achieve the same performance as in the base case. A possible approach would be to add iterations to transmit the required packet sizes so that the V_a generated would be the same as the DV_a.

In the worst cases the number of iterations would be $\left(\lceil \log_2(k) \rceil - 1\right) - 1$.

In the rest of the cases, the necessary additional iterations are within 1 and $\left(\lceil \log_2(k) \rceil - 1\right) - 2$.

I	N	n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	1	0															
2	3	2	1	0	2													
2	4	3	2	1	0													
3	5	4	3	2	1	0	4	3	2									
3	6	5	4	3	2	1	0	5	4									
3	7	6	5	4	3	2	1	0	6									
3	8	7	6	5	4	3	2	1	0									
4	9	8	7	6	5	4	3	2	1	0	8	7	6	5	4	3	2	
4	10	9	8	7	6	5	4	3	2	1	0	9	8	7	6	5	4	
4	11	10	9	8	7	6	5	4	3	2	1	0	10	9	8	7	6	
4	12	11	10	9	8	7	6	5	4	3	2	1	0	11	10	9	8	
4	13	12	11	10	9	8	7	6	5	4	3	2	1	0	12	11	10	
4	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	13	12	
4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	14	
4	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Fig. 3 *Table of V_a for diferent N*

3.2.5 Final Algorithm

Adding the necessary changes required for all three cases, the final algorithm is as shown in Fig. 2.

```
Function TransmitVoice (VoicePacket myVoice, int numNodes,
int myPosition)
{
  N= numNodes;
  n= myPosition;
  AllPacketReceived.add ( myVoice );

  For (i=1; i < log2(N); i++)  {
    NodeDestination =  n + 2^(i-1);
    NodeOrigin = n - 2^(i-1);

    Parallel   { {
       PacketReceive = receive(NodeOrigin);
       AllPacketReceived.add (  Mix (PacketReceive,
AllPacketReceived [i-1] ) ); }
      { PacketSend =  AllPacketReceived [i-1];
      Send(NodeDestination, PacketSend);}}
  }
  Float  X= log2(N - 2^(i-1))

  If ( x÷⌈x⌉=1 )  {
    NodeDestination =  n + 2^(i-1);
    NodeOrigin = n - 2^(i-1);

    Parallel   { {
       PacketReceive = receive(NodeOrigin);
       AllPacketReceived.add(Mix(PacketReceive,
AllPacketReceived[i-1]));  }
      { PacketSend =  AllPacketReceived [ log2(N - 2^(i-1)) ];
      Send(NodeDestination, PacketSend); }}
  }
Else
    TransmitVoiceLastPackets (VoicePacket myVoice,
int numNodes, int myPosition);
}
```

Fig. 2 *Final Algorithm*

3.3 Results

Observing the table of Fig. 4, we can determine the maximum number of machines that the protocol would allow using the different algorithms described and for several transmission speeds.

In this table we suppose that each machine has the capability of transmitting in full-duplex and symmetrically. In the case of an asymmetric speed transmission channel, we have to choose the lowest speed.

The best case is the maximum number of possible machines if only cases 1 and 2 are allowed; and the worst case is the maximum number of possible

machines allowing all three cases. The server value is the maximum number of machines if a machine capable of that speed was promoted as a centralized server for the rest of nodes.

These values are determined using equations (16), which represents the best case scenario, (17), which represents the worst case scenario, and (14), that represents the server case.

In all cases the aim is that the required time to transmit the data is less than the sampling time. These values are considering only the transmission time.

$$2 \cdot \left(\frac{(n-1) \cdot Tam}{BpsTrans} \right) \leq \frac{(Tam - kAddCif)}{BpsSamp} \quad \textbf{(14)}$$

$$k \cdot \left(\frac{Tam}{BpsTrans} \right) \leq \frac{(Tam - kAddCif)}{BpsSamp} \quad \textbf{(15)}$$

$$n_b = 2^k \quad \textbf{(16)}$$

$$n_w = 2^{\left\lfloor \frac{k+2}{2} \right\rfloor} \quad \textbf{(17)}$$

In equations (14) to (17) we consider the following:

- *n* is the maximum number of machines.
- *k* is the maximum number of iterations.
- *Tam* is the size of the packet to be sent in bytes.
- *kAddCif* is the number of control bytes added by the cipher.
- *BpsTrans* is the transmission speed in bytes per second.
- *BpsSamp* is the sampling rate in bytes per second.

In our implementation, *kAddCif* is fixed to 48 bytes, *BpsM* is 11025 bytes per second and *Tam* is 600 bytes.

Observing the results shown in the table of Fig. 4 we can conclude that with a 1,5 Mbps transmission speed we can obtain adequate results for most cases and that with 256 Kbps or less, we can begin to have a multiconference.

4 Conclusion

In this paper we propose a new original scheme to perform multiconference for a wide range of applications including secure communication.

This scheme is P2P based so it limits the damage caused by a single node failure and is more resistant to denial of service attacks. It is lightweight and scalable and can be implemented on limited resources machines like mobile phones or PDAs.

The proposed algorithm presents excellent multiconferencing features for all possible applications (e. g. distance collaboration systems, on-line meetings or internet gaming).

As future lines of research we plan to introduce dynamic audio quality techniques; and to adapt the protocol to incorporate different audio processing techniques relevant for several future applications.

Speed	Best Case	Worst Case	Server
28,8 Kbps	1	1	1
33,6 Kbps	1	1	1
56 Kbps	1	1	1
64 Kbps	1	1	1
88,2 Kbps	1	1	1
128 Kbps	2	2	1
176,4 Kbps	2	2	1
256 Kbps	4	4	2
384 Kbps	16	8	3
768 Kbps	256	32	5
1,5 Mbps	32768	256	8
2 Mbps	1048576	2048	11
4 Mbps	2,199E+12	2097152	21
6 Mbps	4,6117E+18	4294967296	32

Fig. 4 *Table of N for different transmission speeds*

References:

[1] ITU, *H.323v5*, ITU, 2003

[2] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron, SCRIBE: A large-scale and decentralized application-level multicast infrastructure, *IEEE JSAC*, 2002.

[3] S. Deering, Multicast routing in internetworks and extended lans, *Proc. of ACM SIGCOMM*, 1988.

[4] Y. h. Chu, S. G. Rao, S. Seshan, and H. Zhang., Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture., *Proc. of ACM SIGCOMM*, 2001.

[5] Xiaohui Gu, Zhen Wen, Philip S. Yu, ZonYin Shae, Supporting MultiParty VoiceOverIP Services with PeertoPeer Stream Processing, *Proceedings of the 13th annual ACM international conference on Multimedia*, 2005, pp. 303 - 306.

[6] M. Handley, H. Schulzrinne, E. Schooler, J Rosenberg, *Session Initiation Protocol (SIP)*,The Internet Society 1999

[7] Salman A., Baset and Henning Schulzrinne, An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. *Technical Report CUCS-039-04*, Columbia University, 2004