Phone Server:
Design, Implementation and Performance Evaluation

Bo Yang
Distributed Systems Group
Computer Science Department
Stanford University
September 2001
1 Introduction
A phone server processes voice calls between its clients over the Internet. Different from the current Internet phone call models, a phone server not only processes the call setup signaling, but also processes all the voice packets between its clients. This model provides features that are hard to support in the existing models, such as call screening, conference calls, support of mobility and phone tapping. In addition, with this model, there is no need for each caller to have a globally unique IP address. A prototype of the phone server is implemented and its performance is evaluated. In particular, the processing latency of voice packet on the phone server and the performance of sender-initiated handoff are studied.

In the prototype implementation, RAT (Robust Audio Tool) [RAT] is used as the audio tool on the clients. The phone server itself is developed from scratch. TCP-RTM (TCP Real Time Mode) [LC01] is used for the TCP connections between the phone server and its clients. The remainder of the report is organized as follows: Section 2 describes the design and implementation of the components of the phone server architecture; section 3 presents the results of the performance evaluation of the phone server; section 4 draws conclusions of the report; section 5 outlines possible future work.

2 Design and Implementation

2.1 Operation Model of a Phone Server
A phone server processes the call setup/teardown signaling and relays voice packets between its clients. This high-level model is illustrated in fig. 1.

![Fig. 1 Abstract Model of Phone Server](image)

To be able to take voice calls through the phone server, a client sets up a TCP connection with the phone server beforehand. This connection represents the "presence" information of a client to the phone server. Future signaling indicating incoming voice calls as well as voice packets can be sent over this connection. To place a call through the phone server, a client opens a TCP connection to the phone server, if one does not exist yet, and then send an OPEN message that includes the identity of the intended callee. Using the presence information, the phone server "locates" the callee, relays this OPEN message to
the callee over the connection that has been set up beforehand by the callee. If the callee
wishes to take the call, it sends an ACCEPT message to the phone server. Future voice
packets and signaling of this call are relayed by the phone server between the TCP
connections from the caller and the callee.

The Phone Server model provides features that are hard or impossible to implement using
the current Internet phone model, in which call setup is done through the call manager
and all the following processing is done over the direct phone-to-phone connection.
These features are:
- Call screening;
- Conference calls;
- Support for mobility;
- Phone tapping;
- No need for each caller to have a globally unique IP address.

To support voice traffic that has restricted timing requirement, TCP-RTM (TCP Real
Time Mode) [LC01], is used for the TCP connections between the phone server and the
clients. In our prototype implementation, RAT (Robust Audio Tool) [RAT] is used on the
client side as the audio tool to make calls. The phone server itself is developed from
scratch.

2.2 Overview of TCP-RTM

TCP-RTM uses three techniques to make TCP perform well for real-time applications.
First, the receiver is allowed to skip certain lost packets in its receive queue; Second,
when a lost packet is skipped is controlled by a minimum buffer delay calculated by
RTM; Third, the receiver generates heartbeat packets when necessary to keep the flow
going.

To make RTM conform to the congestion control principle, both the sender and the
receiver should support the TCP SNACK (Selective Negative ACK) extension.

Skipping holes in receive queue on the receiver side
Two types of events can trigger the sender to skip holes in its receive queue: when an
out-of-order packet arrives and a process is waiting on this socket to read data; or when a
process tries to read data from the socket, but only out-of-order packet is available. When
some holes in the receive queue are skipped, the receiver acks the sequence number of its
updated rcv_nxt. The information about the skipped holes is contained in SACK.

Minimum Playback Buffer Delay Required by RTM

In-order packets are read from the TCP receive queue as soon as possible. A lost packet
is skipped, if possible, when its playout time comes but it still does not arrive. The
playout time for each packet in the stream is calculated by the receiver based chiefly on
the jitter. To give TCP enough time to let Fast Retransmit to recover lost packets, this
playout buffer delay should be no shorter than a minimum value, $P_{bd_{min}}$, which is calculated using the following formula:

$$P_{bd_{min}} = \frac{3}{2} \times RTT + 3 \times p + \delta$$  \hspace{1cm} (1)

In this formula, RTT is the round-trip time, $p$ is the interval between each packet, and $\delta$ is some processing overhead. Fig. 2 shows that this $P_{bd_{min}}$ value is the minimum needed for Fast Retransmit to take effect.

![Fig. 2 Playback delay buffer needed for Fast Retransmission](image)

**Application level heartbeat**

Allowing the receiver to skip certain packets does not help sustain TCP's performance when the packet loss ratio is high. The sender constantly gets stuck in doing retransmission for a lost packet. This significantly slows down the advancement of the sending window, making a lot of packets obsolete even before they are sent out. To reduce the impact of this type of slowdown, heartbeat is introduced. The ACK information carried by these messages help the sender to move its sending window. In the TCP-RTM implementation [LC01], a heartbeat messages is generated when the number of bytes in the receive buffer drops below a certain threshold.

**2.3 Overview of RAT**

In our prototype implementation, the Robust Audio Tool [RAT] is used as the audio tool to place voice calls. RAT is an open-source audio conferencing and streaming application that allows users to participate in audio conferences over the internet. These can be between two participants directly, or between a group of participants on a common multicast group.
RAT consists of three major components, namely the RTP engine, the GUI engine and a main controller. These components communicate with each other using MBus. The networking module in the original RTP engine runs only on top of UDP. The Mbus module also uses UDP to send and receive its messages.

2.4 Architecture of the Phone Server

The phone server is a single process that accepts connection requests from the clients and relays packets between connections that belong to the same voice call session. All socket I/Os are non-blocking. The main task loop is as follows:

```c
while (1) {
    accept new connection requests;
    select() on open sockets and relay packets
}
```

In TCP-RTM, user level applications inform TCP to skip lost packet by issuing the system call read() at the correct time to read from the socket. In order to support RTM, on the phone server each voice stream keeps track of its own timeline and determines when a read() call should be issued. To make the implementation efficient, a mechanism similar to polling is used rather than having each voice stream setting its own timeout. In this polling implementation, the phone server does a check on all the active voice streams with a fixed frequency (every 10 ms in the prototype implementation). In the check, each voice stream updates its timeline. If the expiration time for the next expected packet occurs and that packet has not been received, the voice stream issues the read() call to let TCP skip that packet.

The buffer delay $D$ for the packets is recomputed at the beginning of each talkspurt. The calculation is based on the $Pbd_{min}$ of RTM as shown in (1). Since at the arrival of a packet, a time period of one-way delay, which is roughly $1/2 \times RTT$, has already passed, $D$ is actually calculated as the following:

$$D = RTT + 3 \times p + \delta$$  \hspace{1cm} (2)

Therefore, the expiration time point $t_{expire0}$ for the starting packet $P_0$ of a talk spurt is computed as:

$$t_{expire0} = t_{now} + RTT + 3 \times p + \delta$$  \hspace{1cm} (3)

in which $t_{now}$ is the time at which $P_0$ arrives.

The expiration time point for the packets in the rest of the talk spurt is calculated as

$$t_{expire(n+1)} = t_{expire(n)} + p$$  \hspace{1cm} (4)
in which p is the interval between packets. To get the RTT value, a TCP socket option is added to expose the RTT measured by TCP to the applications. This option, together with other ones, is listed in table 1.

The start of a talk spurt is detected by several means: 1) marker bit in the RTP header; 2) RTP sequence number and timestamp range mismatch between previous and current packets; 3) a big change in delay estimate; 4) payload type change. The end of a talk spurt is detected by experiencing an extended period of "silence", during which no packet arrives.

In-order packets are read in as soon as possible. The select() call detects arrivals of in-order packets, which are then read in immediately (the read here needs to consider application level framing, which is discussed in <section 2.6>). The read() call upon expiration time is only used when the voice stream gets stuck with a lost packet.

The use of the buffer delay on the phone server, together with the buffer delay on the receiver and RTT etc., might make the overall one way delay larger than some maximum values, say 250 ms, that is required for interactive applications. If this happens, the phone server needs to make a decision as to whether to stop using the buffer delay. If the network loss rate on the incoming connection's link is very low, stop using the buffer delay is not a problem. However, if the network loss rate on the incoming connection’s link is relatively high, the phone server needs to make a tradeoff between high packet loss rate (if stop using the buffer delay) and excessive long delay (if keep the buffer delay). In this case, if the network loss rate is higher on the incoming connection than the outgoing connection, the buffer delay is kept. Otherwise, the phone server can stop using the buffer delay.

To estimate the network loss rate, some TCP socket options are added to let TCP expose certain network statistics to the applications. These options and their meanings are explained in table 1.

<table>
<thead>
<tr>
<th>socket option name</th>
<th>meanings</th>
<th>on TCP sender or receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP_RCVR_3DUP_ACK</td>
<td>a count of 3-dup acks the receivers sends out and thus an estimate of how many fast retransmits are triggered on the sender side</td>
<td>receiver</td>
</tr>
<tr>
<td>TCP_RCVR_SKIP_PKT</td>
<td>number of lost packets skipped by RTM</td>
<td>receiver</td>
</tr>
<tr>
<td>TCP_SNDR_FAST_RETX</td>
<td>number of fast retransmit triggered</td>
<td>sender</td>
</tr>
<tr>
<td>TCP_SNDR_RETX_TOUT</td>
<td>number of retransmission timeout</td>
<td>sender</td>
</tr>
<tr>
<td>TCP_RTT</td>
<td>round trip time measured by TCP</td>
<td>sender and receiver</td>
</tr>
</tbody>
</table>

The network loss rate of the incoming connection (phone server as the receiver) is estimated as the following:

\[
\text{network loss rate} = \text{TCP\_RCVR\_3DUP\_ACK} + \text{TCP\_RCVR\_SKIP\_PKT} \quad (5)
\]

The reason to include TCP_RCVR_SKIP_PKT is because certain lost packet does not trigger a fast retransmit.
The network loss rate of the outgoing connection (phone server as the sender) is estimated as:
\[
\text{network loss rate} = \text{TCP\_SNDR\_FAST\_RETX} + \text{TCP\_SNDR\_RETX\_TOUT} \quad (6)
\]
This is simply the sum of the number of fast retransmissions and the number of retransmission timeouts.

### 2.5 Extensions to RAT to Support TCP-RTM

The original RAT implementation uses UDP, not TCP. Extension to RAT is added to make it work over TCP. This is achieved by adding some relevant TCP functions and a wrapper to the existing networking functions in RAT. The wrapper determines if UDP or TCP should be used. Modules of RAT communicate with each other using Message Bus (Mbus), which runs on top of UDP. UDP is still used for these Mbus calls. For the "real" networking calls, TCP is used instead. Fig. 3 illustrates the extension to RAT to support TCP.

![Fig. 3 Extension to RAT to Support TCP. The major modules of RAT are GUI controller, main controller, and the RTP engine. Mbus uses UDP. Network connections use TCP.](image)

To support RTM, RAT needs to be modified to skip frames that have not arrived by their playout time point. Also, the playout calculation algorithm of RAT needs to be modified to consider the requirement of the minimum playback buffer delay of RTM.

The playout buffer calculation on RAT is based on the one described in [RKTS94]. The buffer delay is computed at the beginning of each talkspurt. Expressed using the playout time point, if \( P_i \) is the first packet of a talkspurt, its playout time, \( p_i \) is computed as
\[
pi = ti + d + 3 \times v, \quad (7)
\]
where $d_i$ and $v_i$ are estimates of the mean and variation of the end-to-end delay during the talk spurt, $t_i$ is the timestamp put on this frame by the sender. The $d$ here includes the clock difference between the sender and the receiver.

The playout point for any subsequent packet in a talkspurt is computed as an offset from the point in time when the first packet in that talkspurt was played out. If $i$ was the first packet in a talkspurt and packet $j$ belongs to this talkspurt, the playout point for $j$ is computed as:

$$p_j = p_i + t_j - t_i. \quad (8)$$

To impose RTM's minimum playback buffer requirement, $p_i$ for the first packet in the talkspurt should be calculated as

$$p_i = \max( (t_i + d + 3 \cdot v), (t_i + 3/2 \cdot RTT + 3 \cdot p + delta) )$$

However, since the $d$ part in $(t_i + d + 3 \cdot v)$ includes the clock difference between the sender and the receiver, we cannot compare $(t_i + d + 3 \cdot v)$ and $(3/2 \cdot RTT + 3 \cdot p + delta)$ directly. Rather, since $d$ represents a one-way delay which is roughly $1/2 \cdot RTT$, we could compare $(3 \cdot v)$ and $((3/2 - 1/2) \cdot RTT + 3 \cdot p + delta)$ instead. Therefore, the calculation of the playout point of $p_i$ becomes:

$$p_i = t_i + d + \max( (3 \cdot v), (RTT + 3 \cdot p + delta) ) \quad (9)$$

With the value of $p$ as 20 ms and a value of $v$ of less than 20 ms, this means the new playout buffer delay is almost always larger than the one calculated using (1). How much extra delay is introduced depends on the value of RTT and the difference of $v$ and $p$.

To make RAT skip expired packets, the current timeline is calculated each time after in-order packets are read in from the socket. If the frames in the application buffer so far fall behind the timeline, RAT will try to skip the lost packet and read in out-of-order packets to keep up with the timeline.

### 2.6 Support of Application-Level Framing

TCP-RTM is allowed to drop packets. At the same time, it is possible for TCP to have multiple packets combined or a single packet broken up into separate ones. The result is the loss of the application-level framing and the incorrect interpretation of a frame on the receiver. To remedy this, we implemented the following scheme with modifications to TCP and user applications.

First, TCP-RTM is modified to preserve the application-level framing. In the modified TCP-RTM, user-level frames are kept as they are in TCP level. Collapsing of two frames and fragmentation of a frame are disallowed. It is the application's responsibility to make sure each frame does not exceed the maximum segment size. About 15 lines of new code are needed for this change.
Second, a size field is added to each application level frame. This information is used by
the receiver for a further check on the sizing of each frame and facilitates the read. This is
implemented using the extension option of the RTP header [SCFJ00]. The format of this
extension is: type (type of extension, 2 bytes), length (length of extension, 2 bytes), and
content (size of the packet, 4 bytes). This is shown in fig. 4.

The phone server and the RAT receiver both use this size information to decide how
much they should read in each time.

3 Performance Evaluation of Phone Server

3.1 Processing Latency of Phone Server

The phone server relays packets between TCP connections that belong to the same voice
call. A frame is received from the caller connection's socket and then written to the socket
of the callee's connection. We expect that this overhead should not be significant enough
to have a severe impact on the quality of the voice call. Experiments were conducted to
verify this.

In the following experiments, a client running RAT sends an audio stream to a receiver,
which is also running RAT, through a phone server. One audio stream sends out a 320-
byte packet every 20 ms, which corresponds to a bandwidth of 128 kbps. Packets going
through the phone server were tracked by tcpdump. The time when the packet first
arrived and finally was sent out were recorded. Each packet is identified by its RTP
sequence number. The time difference between the kernel first and last sees the same
packet is computed and this difference corresponds to the processing overhead of the
packet by the phone server.
To test the scalability of the phone server, background traffic that also goes through the phone server is generated between a second pair of clients. Each background connection mimics one audio stream used by RAT, sending out a 320-byte packet every 20 ms. The phone server relays the background traffic in the same way as it does for the audio stream of RAT. Different work load on the phone server is simulated by generating different number of background connections.

The experiments were conducted over 100 Mbps Ethernet. The two clients are each running a 933 MHz Pentium III processor with 256 MB of RAM. The phone server runs a 600 MHz Pentium III processor with 512MB of RAM. The results are shown in table 2. The processing latency on the phone server increases from 2.22 ms to 12.8 ms when the number of background connections increase from 0 to 100. Overall, the latency does not introduce too much overhead, considering the maximum one-way delay tolerated by voice calls is 250 ms.

<table>
<thead>
<tr>
<th>Number of Background Traffic</th>
<th>Total Bandwidth of Background Traffic (Mbps)</th>
<th>Average Latency (ms)</th>
<th>Min Latency (ms)</th>
<th>Max Latency (ms)</th>
<th># of Packets tracked</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>2.22</td>
<td>0.063</td>
<td>15.9</td>
<td>24080</td>
</tr>
<tr>
<td>10</td>
<td>1.3</td>
<td>2.37</td>
<td>0.069</td>
<td>16.2</td>
<td>15715</td>
</tr>
<tr>
<td>50</td>
<td>6.4</td>
<td>2.94</td>
<td>0.108</td>
<td>19.2</td>
<td>18023</td>
</tr>
<tr>
<td>80</td>
<td>10.2</td>
<td>3.91</td>
<td>0.133</td>
<td>20.7</td>
<td>17837</td>
</tr>
<tr>
<td>100</td>
<td>12.8</td>
<td>5.46</td>
<td>0.068</td>
<td>25.9</td>
<td>21396</td>
</tr>
</tbody>
</table>

### 3.2 Performance of Handoff

In the handoff experiment, the sender sends a 128 kbps audio stream through the phone server to the receiver. After a certain time period, the sender does a handoff. During the handoff process, the sender first opens a new connection to the phone server, then switches to this new connection and closes the old connection.

The performance of the handoff is evaluated by three methods. First, the average jitter and the number of late frames experienced by the receiver during the handoff are measured and compared with those of control experiments that do not perform handoff; second, the delay and jitter during the handoff are traced and plotted; third, the quality of the audio playback during handoff is evaluated by listening at the receiver side while handoff is performed by the sender. Results of these three methods are presented in the following sections.

**Average Jitter and Number of Late Frames during Handoff**

The average jitter and late frames experienced by the receiver during handoffs was measured and compared with those of the runs not performing handoffs. The results are shown in table 3. In the experiments, RTT between the sender and the phone server varies from 1 ms to 70 ms. RTT between the phone server and the receiver is 150 usec.
For each RTT values, test runs doing handoffs and not doing handoffs were done for comparisons. For tests with handoffs, two handoffs were performed during a 15 second time period. Each test was repeated three times and the results were averaged.

The results show that no late frames are introduced by performing handoff. For all these RTT values, the average jitter with handoffs happening is slightly larger than that without handoffs.

<table>
<thead>
<tr>
<th>RTT (ms)</th>
<th>handoff or no handoff</th>
<th>Average Jitter (ms)</th>
<th>Total Number Frames Sent</th>
<th>Number of Late Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>no handoff</td>
<td>4.7</td>
<td>4652</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>handoff</td>
<td>4.8</td>
<td>4834</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>no handoff</td>
<td>7.4</td>
<td>4582</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>handoff</td>
<td>8.2</td>
<td>4496</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>no handoff</td>
<td>14.1</td>
<td>4431</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>handoff</td>
<td>14.7</td>
<td>4584</td>
<td>0</td>
</tr>
<tr>
<td>70</td>
<td>no handoff</td>
<td>15.7</td>
<td>4762</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>handoff</td>
<td>15.9</td>
<td>4578</td>
<td>0</td>
</tr>
</tbody>
</table>

Trace of Delay and Jitter during Handoff
The second way, tracing the delay and jitter change during the handoff, revealed a bug in the initial implementation of handoff. After this bug was fixed, the trace did not show obvious delay or jitter increase when handoff occurs.

These results are shown in fig. 5 to fig. 10. Fig. 5 and 6 show the impact of the bug in the initial implementation of handoff. In this initial implementation, connect() calls were done in a blocking manner. Therefore, when the round trip time is large, the sender is blocked, a backlog of audio frames is built up and their delivery is delayed. The result is a sudden increase of delay and jitter when handoff occurs. Fig. 7 and 8 show the trace of delays and jitters when handoff occurred after the bug is fixed. Two handoffs were performed during the test. There was a slight increase of delay and jitter after the second handoff, but not after the first. For comparison, Fig. 9 and 10 show the trace of delay and jitter for a session without handoff.
Fig 5. RTT between the sender and the phone server is 70 ms. RTT between the phone server and the receiver is 150 usec. The network loss rate is 0%. Two handoffs were performed during the test. The arrows on the X axis indicate when the handoff occurred. A sudden increase of the delay when the handoff happens is because of the blocking connect(), which introduces backlog of packets on the sender and consequently longer delay.

Fig 6. This is the same test run as described in fig. 9. RTT between the sender and the phone server is 70 ms. RTT between the phone server and the receiver is 150 usec. The network loss rate is 0%. Two handoffs were performed during the test. The arrows on the X axis indicate when the handoff occurred. This figure shows the change in jitter during the test. Due to the blocking connect() call, a sudden increase of jitter is observed when handoff occurs.
Fig 7. RTT between the sender and the phone server is 70 ms. RTT between the sender and the phone server is 70 ms. RTT between the phone server and the receiver is 150 usec. Network packet rate is 0%. non-blockingly connect() is used to set up the new connection. The two handoff points are marked by arrows on the X axis. No obvious delay increase after the first handoff. Some increase of delay is observed after the second handoff.

Fig 8. This is the same test run as described in fig. 7. RTT between the sender and the phone server is 70 ms. RTT between the sender and the phone server is 70 ms. RTT between the phone server and the receiver is 150 usec. Network packet rate is 0%. non-blockingly connect() is used to set up the new connection. The two handoff points are marked by arrows on the X axis. This figure shows the jitter during the test. A slight increase in jitter is observed after the second handoff.
Fig 9. RTT between the sender and the phone server is 70 ms. RTT between the phone server and the receiver is 150 usec. Network packet rate is 0%. No handoff is performed during the test. This is shown as a comparison to fig. 7.

Fig 10. This shows the jitters during the same test run as described in fig. 9. RTT between the sender and the phone server is 70 ms. RTT between the phone server and the receiver is 150 usec. Network packet rate is 0%. No handoff is performed during the test. This is shown as a comparison to fig. 8.
Quality of Audio Playback during Handoff

Listening to the audio playback did not reveal any perceivable gap during the playback of the audio.

3.3 Quality of Audio Playback

In this test, the sender sends an audio stream to a receiver through a phone server. The playback quality of the audio stream is judged by listening to the playback of the audio at the receiver. Audio streams with different sending rates are tested under different round-trip times and network loss rates. When the network loss rate is very low, the playback is always good even when the sending rate of the audio stream is high. When the loss rate is high and the round-trip time is long, only audio streams with a lower sending rate can provide a satisfiable playback (this is when TCP-RTM is in use. With only regular TCP, the playback simply does not work with an round-trip time of 70 ms and a network loss rate of 5%). Table 4 and 5 show the results of two representative cases.

Table 4. Playback quality with RTT = 70 ms and network loss rate = 0%

<table>
<thead>
<tr>
<th>Audio Stream</th>
<th>Characteristics</th>
<th>Sampling Frequency (kHz)</th>
<th>Channel</th>
<th>Packet Interval (ms)</th>
<th>Packet Size (Bytes)</th>
<th>Bandwidth (kbps)</th>
<th>Playback Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Mono</td>
<td>8</td>
<td>20</td>
<td>320</td>
<td>128</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>8</td>
<td>Stereo</td>
<td>8</td>
<td>20</td>
<td>640</td>
<td>256</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>16</td>
<td>Mono</td>
<td>16</td>
<td>10</td>
<td>320</td>
<td>256</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>16</td>
<td>Stereo</td>
<td>16</td>
<td>10</td>
<td>640</td>
<td>512</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>32</td>
<td>Mono</td>
<td>32</td>
<td>5</td>
<td>320</td>
<td>512</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>32</td>
<td>Stereo</td>
<td>32</td>
<td>5</td>
<td>640</td>
<td>1024</td>
<td>Good</td>
<td>Good</td>
</tr>
</tbody>
</table>

Good - Playback is continuous. No perceivable gap.
Satisfiable - Have gaps during playback. But does not affect understanding of the audio.
Bad - long gaps during playback. cannot understand the meaning of the audio.

Table 5. Playback quality with RTT = 70 ms and network loss rate = 5%

<table>
<thead>
<tr>
<th>Audio Stream</th>
<th>Characteristics</th>
<th>Sampling Frequency (kHz)</th>
<th>Channel</th>
<th>Packet Interval (ms)</th>
<th>Packet Size (Bytes)</th>
<th>Bandwidth (kbps)</th>
<th>Playback Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Mono</td>
<td>8</td>
<td>20</td>
<td>320</td>
<td>128</td>
<td>Satisfiable</td>
<td>Satisfiable</td>
</tr>
<tr>
<td>8</td>
<td>Stereo</td>
<td>8</td>
<td>20</td>
<td>640</td>
<td>256</td>
<td>Satisfiable</td>
<td>Satisfiable</td>
</tr>
<tr>
<td>16</td>
<td>Mono</td>
<td>16</td>
<td>10</td>
<td>320</td>
<td>256</td>
<td>Satisfiable</td>
<td>Satisfiable</td>
</tr>
<tr>
<td>16</td>
<td>Stereo</td>
<td>16</td>
<td>10</td>
<td>640</td>
<td>512</td>
<td>Bad</td>
<td>Satisfiable</td>
</tr>
<tr>
<td>32</td>
<td>Mono</td>
<td>32</td>
<td>5</td>
<td>320</td>
<td>512</td>
<td>Bad</td>
<td>Satisfiable</td>
</tr>
<tr>
<td>32</td>
<td>Stereo</td>
<td>32</td>
<td>5</td>
<td>640</td>
<td>1024</td>
<td>Bad</td>
<td>Bad</td>
</tr>
</tbody>
</table>

Good - Playback is continuous. No perceivable gap.
Satisfiable - Have gaps during playback. But does not affect understanding of the audio.
Bad - long gaps during playback. cannot understand the meaning of the audio.

4 Conclusion

The operation model of the phone server differs from the current models for voice calls over the Internet. A phone server not only processes the call setup signaling, but also
processes all the voice packets between its clients. This model provides features that are hard to support in the existing models, such as call screening, conference calls, support of mobility and phone tapping.

A phone server prototype is implemented and its performance evaluated. The processing latency of voice packets on a phone server is shown to be in the order of several milliseconds, which does not introduce too much overhead, assuming a 250 ms maximum one-way delay for voice calls. The performance of handoff is also evaluated. The results show that sender-initiated handoff through the phone server does not affect the playback quality on the receiver side. Although handoff results in a slightly larger jitter experienced by the receiver, this increase is only within one millisecond and should be easily absorbed by the playback delay buffer on the receiver.

TCP-RTM is used for TCP connections between the phone server and its clients. Support of TCP-RTM is implemented on the phone server and RAT. The effectiveness of RTM is demonstrated by the fact that satisfiable audio playback can be supported when the network loss rate is 5% and the round trip time is 70 ms, which is hard to achieve with regular TCP.

During the implementation of the phone server, support of application-level framing in TCP-RTM is studied. A modification of TCP to let TCP preserve application level framing is implemented. This approach is demonstrated to be effective and adequate in solving the general framing issues for certain TCP variants such as RTM that allows skipping packets.

5 Future Work
The sender-initiated handoff through the phone server can be extended to make a smoother handoff possible. Possible enhancements include "warming up" the backup connection before switching to it and setting up multiple backup connections and selecting the best from them before switching.

The current implementation of the phone server does not support high-level call signaling processing. Features such as call screening and conference calls can be developed.

6 References
[MB00] Biswaroop Mukherjee and Tim Brecht, Time-lined TCP for the TCP-friendly Delivery of