Efficient Voice Over Wireless Network

Author: Wei Zhang
Supervisor: Tim Moors

Abstract
The objective of this project is to make efficient voice transmission over wireless network. To achieve this goal, we changed the data unit that is sent down the protocol stack at the source node from packets to “trickles”, in order to reduce transmission overheads, minimise packetisation and synchronisation delays.

Background
VoIP application samples audio signals, encodes them into audio frames, then transmits them over a network. However, each packet has a certain transmission overhead and often it is much bigger than an audio frame even it is carrying multiple audio samples. For example, in 802.11 WLAN, an 33B audio frame taking 24 us requires at least 145 us of overheads.

Also audio compression occurs asynchronously with respect to network transmission opportunities, so that an audio packet once created may have to wait some time until there is a network transmission opportunity. This also leads to another source delay.

Therefore, to make efficient voice over wireless network, we have to try to reduce transmission overheads and minimise source delays.

What’s new ? - “TRICKLES”
Trickles are a new source data unit that minimises source packetisation and synchronisation delays. In terms of data units, the idea of a trickle is to append new application content to the end of existing content that is still awaiting transmission, and update the headers as it goes down the protocol stack, so that it shares the same overheads.

If we have frequent transmission opportunities, then short packets will be sent. However, if the transmission opportunities are less frequent, more application content will accumulate, and be aggregated for transmission in one packet when the opportunity comes.

How we did it
To implement Trickles, we modified the Linux kernel. We introduced new socket options which provides communication between user application and kernel. We added a new field into the socket structure that keeps track of the last packet that has been put into the device transmission queue. We also modified the “send” function in each protocol stack inside the kernel to check whether there is a waiting packet, and if so, concatenate the new content to it and update the headers.

What was hard?
Updating the header information is a hard task. We have to update IP, UDP and MAC headers when doing concatenation. We also have to figure out a best way of appending new content and updating the headers so that it will take much less time than packetisation delay. This have to be done really carefully. If either of these headers are incorrectly updated, then the concatenated packet will be treated as damaged packet and be discarded at destination.

How good is it?
Performance testing was carried out on both the traditional and Trickles implementation using the voice application “Linphone”. The result is shown in the following table:

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Traditional</th>
<th>Trickles</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-to-End Delay</td>
<td>21 ms</td>
<td>0.5 ms</td>
</tr>
</tbody>
</table>

The end to end delay is the amount of time that passes between when new content is sent by the source application and when it is received by the target application.

What is lost?
Appending new content to the end of old content requires additional memory access, since the application will naturally do this itself.

Conclusion
“Trickles” has been successfully implemented in Linux kernel to reduce transmission overheads, minimise packetisation and synchronisation delays, and hence improves efficiency of voice transmission over wireless networks. In the future, we will be keep refining our implementation of this idea to improve its performance.