Using OLSR for Streaming Video in 802.11 Ad Hoc Networks to Save Bandwidth

Elsa Macías, *Member, IAENG*, Alvaro Suárez, *Member, IAENG*, J. Martín and Vaidy Sunderam*

Abstract

Mobile ad hoc networks are prone by nature to path breaks and reconnections. Routing protocols such as OLSR that provide topology information with a minimum delay to the quick reconfiguration of path breaks are desired. Applications and protocols should be able to adapt to the dynamics of these networks. However, this is not true for most applications and protocols. For example, a video streaming server does not become aware of the path break because it does not interact with lower layers that inform about this event. As a result, important resources such as battery and bandwidth are not used efficiently, and TCP connection failures happen frequently. In this paper, we present a proxy based solution to detect path breaks and reconnections using the proactive OLSR protocol for ongoing streaming sessions and we do corrective actions at the application layer to lead a more efficient usage of the bandwidth during disconnections.

Keywords: WiFi ad hoc networks, Streaming the video, Save the bandwidth and battery, Frames lost, OLSR

1 Introduction

Streaming media is a technique that allows the consumption of the media while it is being delivered. It is very useful for devices with low storage capacity such as mobile phones and PDAs and for users connected to WiFi links [1].

Usually *User Datagram Protocol* (*UDP*) is used to transmit live streaming video [2]. The couple *Real-time Transport Protocol* (*RTP*) and *Real-time Transport Control Protocol* (*RTCP*) [3] are used to transmit real time streaming video [4]. The server continuously sends frames and the client can not pause the streaming. The persistent version of *HiperText Transfer Protocol* (*HTTP*) [5] also supports streaming for *Video On Demand* (*VoD*) so a client sends a request and gets a response, and then sends additional requests and gets additional responses without *Transmission Control Protocol* (*TCP*) connection release. In [6] a new video file format is presented to achieve this. A standard protocol that allows the user remotely control a streaming media server using commands such as play and pause defining a streaming session is *Real Time Streaming Protocol* (*RTSP*) [7]. RTSP can use any of the above protocols for transmitting multimedia data.

To manage efficiently the disconnections and reconnections of participating wireless devices in the streaming session, some streaming servers set up a socket for verifying peer host accessibility to avoid the maintenance of a session indefinitely open (following the polling recommendation of [8]). After several unsuccessful requests, the server silently frees the software resources reserved for serving the client and the session ends. A new attempt of the reconnected client to continue with the session will be not accepted, that is to say, the client and the server must set up a new session and the content must start from the beginning. A design parameter is the number of unsuccessful requests considered before closing the session. As far as we know, this is chosen irrespective of a wireless or a

*Manuscript received August 31, 2006. An earlier version of this paper was published in IWWN'06 (IMECS'06): "Using OLSR for Seamless Streaming Video in 802.11 Ad Hoc Networks", pp. 932–937, Hong Kong, 2006. Research partially supported by the Spanish CICYT (MEC) and European Regional Development Fund (FEDER) under Contract TSI2005-07764-C02-01 and The Canaries Regional Education, Cultural and Sports Ministry and FEDER under Contract PI042004/164. Elsa M. Macías and Alvaro Suárez are with Grupo de Arquitectura y Concurrencia, Departamento de Ingeniería Telemática, Universidad de Las Palmas de G. C. (ULPGC), Campus Universitario de Tafira, 35017 Las Palmas de G.C., Spain. Tel: 34 928 451239; Fax: 34 928 451380; Email: {emacias,asuarez}@dit.ulpgc.es. J. Martín is a graduate of Technical Engineering of Telecommunications at ULPGC. Vaidy Sunderam is with Department of Math and Computer Science, Emory University, Atlanta, USA. Email: vss@mathcs.emory.edu.

(Advance online publication: 13 February 2007)

fixed client is being served. However, it could be desirable to try a higher number of attempts before closing a session with a wireless client due to the problems associated with the mobility, high error rates, hidden terminal and radio signal propagation.

Notice that during the polling, the server continues with the frame transmission and the traffic sent will be lost whenever the client is out of coverage and the transport protocol is UDP. If TCP is used to transport the data, the TCP reliability mechanism will retransmit the missing data. The socket will become invalid to the server or the client if an abort occurs and with high probability the session will end abruptly. Aborts primarily occur when data goes unacknowledged for a period of time that exceeds the limits on retransmission defined by TCP. Other causes for an abort include a request by the application, too many unacknowledged TCP keepalive probes, receipt of a TCP reset packet, and some types of network failures reported by the IP layer.

Some work has been recently done to improve the performance of TCP [9] over ad hoc networks for streaming taking into account mobility-induced disconnections, reconnections and high-of-order delivery ratios. They use a novel multimetric join identification of packets and connection behavior to simulate the performance of their proposal. It is important to control the disconnections and reconnections of wireless devices that participate in the streaming session. A good control, pausing the server while the client is disconnected (or any other device in the network path from the server to the client), it will improve the energy consumption of batteries [10], it will save bandwidth because the data transmission is frozen, and it will avoid the data lost.

In [11] it is presented a proactive adaptation to the UDP-based streaming video sent by a fixed server to a mobile client connected to one 802.11b infrastructure wireless network. The adaptation consists of increasing the buffer size on the client to store more frames just before entering the low quality area (termed *trouble spot*) in the hope that the mobile client will exit the trouble spot before the buffer runs out.

In [12] we presented a proactive mechanism that detects path breaks and reconnections between the video streaming server and the client for a VoD ongoing RTSP session and a solution at the application layer based on proxies that provides the corrective actions during disconnections and reconnections.

In this paper we consider not only UDP-based streaming video but also TCP-based. On the other hand, we consider both the client and the server mobiles in a *mobile ad hoc network* (*MANET*). In this scenario, path breaks take place frequently and quickly due to the movement of end nodes or intermediate nodes in the path (we use *Optimized Link State Routing Protocol* (*OLSR*) [13] for the quick reconfiguration of path breaks for a MANET) so the proactive adaptation proposed in [11] could not be viable.

We present streaming sessions that use RTSP, standalone RTP or HTTP protocols, we include corrective actions during disconnections and reconnections, we manage long disconnections and the TCP connection failures. More in detail, the contributions of this paper are:

- A system to control the client's availability in order to avoid the sending of frames while it is temporarily disconnected.
- A system to announce the UDP services provided by the server in order to let the clients request UDP streams on demand. Otherwise, the server can not control the client's availability due to the connection-less nature of the UDP protocol.
- A mechanism to pause the VoD of ongoing RTSP sessions to save computing resources and battery on the server while the client is disconnected. In this paper we are not concerned about the losing of frames during a disconnection for live video sessions or VoD over HTTP or RTP, i.e, streams that can not be paused with RTSP commands.
- A mechanism for transparently detecting TCP connection failures and to create a new TCP connection that avoids the streaming session release and to end sessions with clients disconnected for a long time that use TCP to transport data or commands (i.e. RTSP and HTTP).
- An analysis of the experimental results that show the benefits of using our whole system.

All the above contributions have been developed without modifying neither the player nor the server, and not even the streaming media protocols. For doing that, we use proxies.

The rest of the paper is organized as follows: section 2 reviews the hardware and software architecture. Section 3 is devoted to present in detail the system to control the client's availability and to announce the UDP services provided by the server. In section 4 is described the proxies' behavior during disconnections and reconnections for different streaming protocols. Section 5 presents the mechanism to manage TCP connection failures and end sessions with clients disconnected for a long time. Then in section 6 we describe our experimental tests. And finally, concluding remarks and ongoing research are summarized in section 7.

2 Hardware and Software Architecture

The topology we consider is shown in Fig.1.a. It consists of a MANET of any number of hops, where the server node (S) communicates with the client node (C) directly or via one or more intermediate nodes (I) due to the movement of end nodes. Fig.1.b to d shows the network architecture for the different nodes. The shaded parts in Fig.1.b to d are the new software elements we introduce to avoid modifying the client, the server and the streaming protocols:

- *olsrd* (OLSR daemon) [14] is an implementation of OLSR. OLSR routes efficiently the packets into the network according to the number of hops between the sender and the receiver. Due to the time-varying characteristics of the wireless links, olsrd can be configured to calculate the optimal routes defined as the number of attempts by a node on average to successfully transmit a packet to a destination, instead of the number of hops. OLSR consists of:
    - A neighbor sensing mechanism that detects changes in its neighborhood injecting and receiving HELLO messages periodically.
    - An efficient flooding of control traffic, i.e. OLSR packets injected into the network for the quick reconfiguration of path breaks. All nodes receive the messages and there are not duplicated messages thanks to the use of multipoint relays (concept invented for HiperLAN/1). This is an important property that favors its use in a wireless network which is by nature prone to mobility of nodes and collisions due to the hidden terminal problem.
    - Diffusion of topological information necessary to obtain optimal routes in terms of the number of hops. This information is valid for a period of time so expired information is removed.

  All the traffic in OLSR is UDP and it is transmitted by broadcast or multicast on port 698.

- The communication between the client and the server is made by proxies. The proxy on the server and client side are called *proxy server* and *client agent* respectively. Since both the client and the server are mobile, the proxy server is installed on the wireless node that serves the stream (S), and the client agent on the client node. The proxies are also in charge of detecting if there is, or is not, a path between the server and client to do corrective actions. These corrective actions depend on the type of video streaming being served (VoD or live video) and the type of streaming protocol used to transport the data (RTSP, HTTP or RTP) as we will show in section 4.

- To control the client's availability and to announce the UDP services we use olsrd as the most efficient way to transport our own control and announcement packets into the MANET. OLSR lets use its optimized flooding mechanism to send information, routing related or not, from the application level using a plug-in. We just use this property to inject user defined packets (OLSR packets type 200) for our purposes as we will describe in next section. The plug-in on the server, client and intermediate nodes conveys to olsrd the information to be sent into OLSR packets type 200 to the MANET. The plug-in on the server and client nodes also communicates to the proxies the OLSR packets type 200 captured by olsrd from the network.

- *TCPControl* is the mechanism for transparently detecting TCP connection failures and to create a new TCP connection that avoids the streaming session release. Despite the fact there are enhancements to TCP for MANET, we do not consider them because of the following reasons: some require modifications to the existing TCP protocol (e.g. TCP-F [15] and Split-TCP [16]), more bandwidth and power consumption during a path failure is required (TCP-ELFN [17]), dependency on a particular routing protocol to improve its performance (TCP-Bus [18]), addition of layers to the TCP/IP protocol stack (ATCP [19]), implementation of a new transport protocol that leads a lack of interoperability with TCP

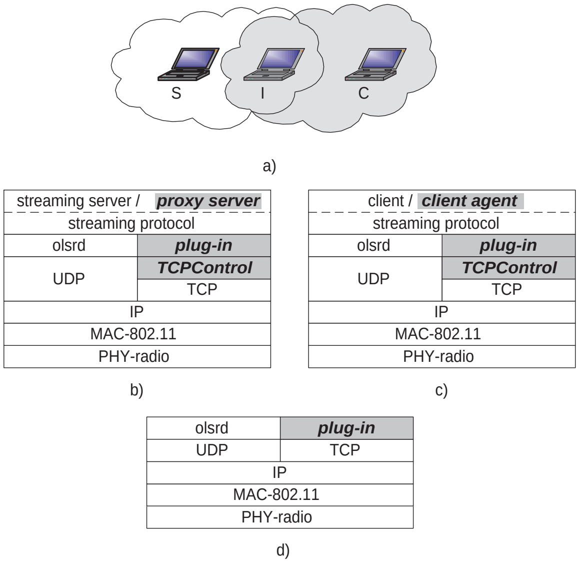

Figure 1: A two hop MANET. Topology (a), Software architecture: server node (b), client node (c), intermediate node (d).

(ACTP [20] and ATP [21]).

3 Detecting Client's Availability and UDP Services Announcement

Both tasks are done injecting and capturing OLSR packets type 200. Whenever the communication between the client and the server is possible (directly or via one or more intermediate nodes), the proxy server receives periodically OLSR packets type 200 from the client agent and viceversa. If the proxy server does not receive at least one of this kind of packet for a while (1 second by default although this timeout can be modified by the user when the proxy is started), this is indicative that the client is disconnected. Similarly, if the client agent does not receive a packet OLSR type 200 from the proxy server (after 1 second by default, also configurable), it becomes aware of the disconnection. The reconnection is detected by the proxy server and the client agent when they receive at least one packet OLSR type 200 from the other one during an interval of 1 second. Appropriate actions are done on both peers when the disconnection or the reconnection are detected and they depend on the type of streaming protocol used. The chosen timeout (1 second in our experiments) is a key parameter because a high value could lead a high delay to detect the disconnection whereas a low value could trigger false alarms, i.e., no packet is received because of network congestion but the proxy server or the client agent wrongly detect a disconnection.

The packet size injected by the proxy server and the client agent is 206 and 66 bytes respectively. The difference in size is due to the message content differs. For example, the proxy server is concerned about broadcasting the type of server (e.g. VideoLan [22]), its IP address, the proxy server's name to the client agent can distinguish among different proxies, a command used by the TCPControl mechanism, the play listing using RTSP, and HTTP and UDP ports available to serve streams. As you can see, the broadcasting of the own OLSR packets type 200 is also the mechanism used to announce the UDP services. The message content is useful since the client agent shows it to the user via a *Graphical Unit Interface* (*GUI*) and then it launches the user's player with the appropriate command according to the user's election. The content of the packet injected by the client agent includes its IP address, the type of player and requests of UDP streams.

The injection (steps *3* or *7* in Fig.2) and capture (steps *4* or *8*) of these packets is done by olsrd. The construction of these packets is done by a plug-in (step *2*) with the information provided by the proxies (step *1*) via a shared file. The proxies update the information to be broadcasted, when necessary, by rewriting this file (e.g. to send a command of the TCPControl mechanism, a new UDP service request and so on) and the plug-in read from this file. By default, ten packets are injected per second. The plug-in on the OLSR intermediate nodes forward the packets type 200 according to the OLSR standard by using the *default forwarding algorithm* [13] (steps *4* to *7*) and they are not passed to the application level. On the contrary, if this packet reaches to the client or server node (step *8*), olsrd passes it to the plug-in (*step 9*) and then the plug-in unwraps it and delivers it to the proxy server only if the packet's source is the client (step *10*). The packet is delivered to the client agent if it contains information about the server (step *10*).

4 Corrective Actions

Table 1 summarizes the actions that the proxies do when they detect disconnections and reconnections (in brackets it is shown the process that does the action), and the benefits of these actions. Fig.3 and Fig.4 show the proxies behavior during disconnections and reconnections for RTSP, and HTTP or RTP respectively. Irrespective of the streaming protocol, the client agent

Table 1 *Corrective actions during disconnections and reconnections.*

	Disconnection	**Reconnection**	**Total disconnection**	**Lost frames**	**Abrupt ending**	**Batt. saving**	**BW saving**
RTSP	Pause the server (PS), warning the user (AC) and close TCP connection (PS,AC)	Create TCP connection (PS,AC), resume the server (PS) and warning the user (AC)	End session (PS)	Yes (live video), No (VoD)	No	Yes	Yes
HTTP	Freeze frames forwarding from PS to AC, close TCP connection (PS,AC) and warning the user (AC)	Create TCP connection (PS,AC), resume frames forwarding (PS) and warning the user (AC)	End session (PS)	Yes	No	No*	Yes
RTP	Freeze frames forwarding from PS to AC and warning the user (AC)	Resume frames forwarding (PS) and warning the user (AC)	End session (PS)	Yes	–	No*	Yes

PS: Proxy Server **AC**: Agent Client **BW**: Bandwidth

*The server is still sending frames but PS does not inject them into the MANET

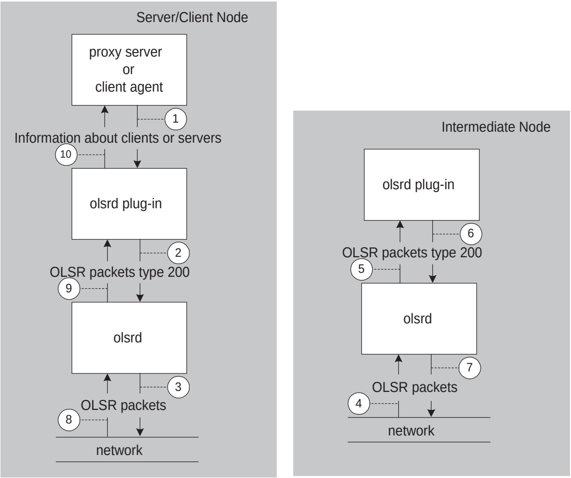

Figure 2: Information passing between proxies.

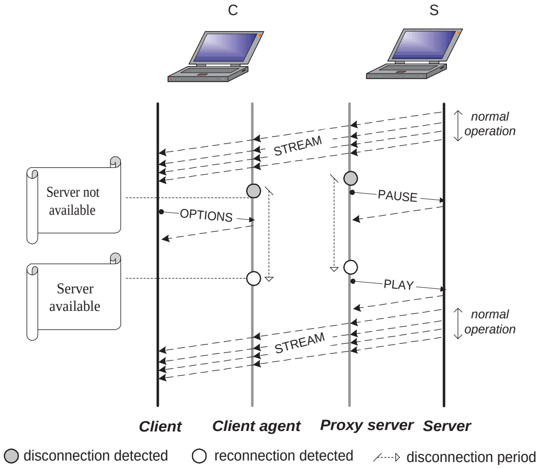

Figure 3: Proxies behavior during disconnections and reconnections for a RTSP session.

starts a warning message box on the user's screen when a disconnection or a reconnection is detected and the proxy server ends the session when the disconnection exceeds a period of time. For the streaming protocols built on top of TCP (RTSP and HTTP), the TCP connection between the proxy server and the client agent is closed when a disconnection happens and a new one is created after the reconnection. Using RTSP compliant commands such as pause and play, the proxy server pauses or resumes the server. For HTTP or RTP, the server is not paused during the disconnection period but the frames are not forwarded from the proxy server to the agent client to save bandwidth.

5 The TCPControl Mechanism

When establishing one TCP connection for a RTSP or HTTP session, three TCP connections are created: one between the client and the client agent, another to communicate the client agent and the proxy server, and the last one for the communication between the proxy server and the server. The former and the latter are local connections and they are not likely to fail during disconnections. The mechanism is as follows (Fig.5):

- *Identify the TCP connection.* After the second of the three TCP connections is created, the client agent requests a unique identifier for this TCP connection by sending a *TCPControl:0* command to the proxy server using the newly TCP connection. The proxy server responds to the client agent with a *TCPControl:id* where *id* is the identifier, a value different for any client agent and the proxy server.

- *Session communication.* The HTTP or RTSP session begins normal communication.

- *Disconnection detected.* If the client is out of coverage, then the client agent do the following actions:

    1. Close the TCP connection with the proxy server.
    2. Detect the server's availability.
    3. Create a new TCP connection with the proxy server.
    4. Send the *TCPControl:id* command to the proxy server to replace the old TCP connection with the last one. The proxy server sends *TCPControl:OK* as acknowledgement. These two commands are communicated using the new TCP connection.

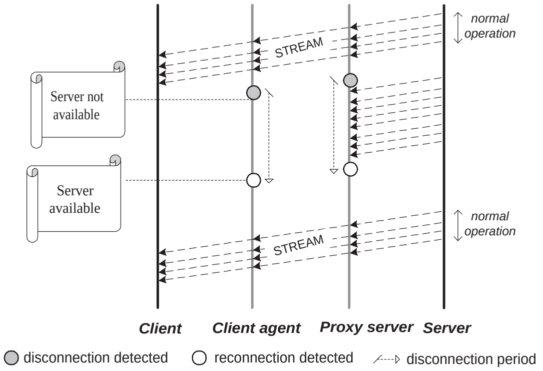

Figure 4: Proxies behavior during disconnections and reconnections for a HTTP or RTP session.

  If the server is out of coverage, it must order all the client agents to close the TCP connections and create new TCP connections. For doing that, the proxy server broadcasts a message using the packet OLSR type 200 including in the message's content of the packet the command *TCPFall* to convey all the client agents that they must do the steps 1 to 4 described in the phase *Disconnection detected.*

- *End sessions with clients disconnected for a long time.* After a timeout, the server frees the software resources reserved for serving the client and the session ends. Any later attempt to recover the TCP connection will fail (command *TCPControl:ERROR*) and in that case the client agent will warn the user that the session has ended.

6 Experimental Tests

We tested the behavior and performance of our software architecture using the topology described in Fig.1.a. We are concerned in presenting results that show: a) the low overhead of the packets OLSR including our packets type 200 in comparison with the data volume of the streaming, b) the low usage of CPU and battery consumption due to the proxies, c) the delay and jitter because of the proxies, and d) the benefits of using our corrective actions and the TCP connections management between the proxy server and the client agent.

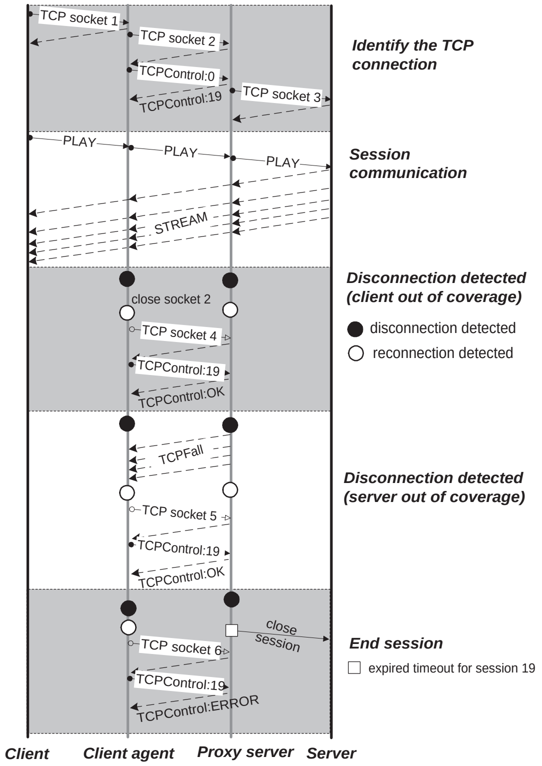

Figure 5: TCPControl mechanism.

The plug-ins and the proxies were programmed using C and C++ languages respectively for Windows operating system. The server was installed on a Pentium IV at 2.8GHz with 512 MB and 802.11b compliant. The intermediate node was a Centrino at 1.6GHz, 512MB and 802.11b/g. The client node was a Celeron 1.4GHz, 1024MB and with a 802.11b/g wireless interface. All the nodes were located in the same room and we added mobility to the network by allowing the client node to be within radio range of the server node via the intermediate node and also we moved the client and the server to make each other out of coverage to test the corrective actions made by proxies and the TCP connections management. We used VLC media player [22], a free cross-platform media player that supports a large number of multimedia formats and it is available for several operating systems, it needs little CPU power and it can be used as a streaming server to stream in unicast or multicast in IPv4 or IPv6. We used also VLC for serving the video in unicast in IPv4.

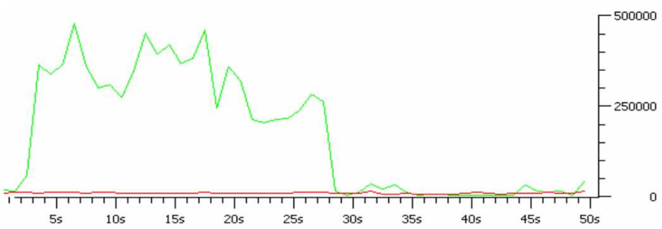

Figure 6: Traffic OLSR (red curve) and RTSP traffic (green curve).

6.1 OLSR Traffic

Fig.6 shows the bytes per second for a RTSP session (green curve) and the traffic due to OLSR protocol including our own packets type 200 (red curve) obtained using Ethereal tool [23]. As you can see, the overhead due to OLSR is negligible in comparison with the traffic of the ongoing streaming session.

6.2 CPU and Battery Consumption

In the experiment, the server sends streams at a rate of 2400Kbps. It was measured the CPU percentage consumed by the proxy server and the client agent for different streaming protocols varying the number of sessions from 1 to 10 (Fig.7.a to c). During the experiment, the processes running on the server were the proxy server, the server and olsrd, whereas on the client were running the client agent, the player and olsrd. As you can see, the proxies need little CPU power being something higher for RTP sessions on the client agent in comparison with the same session on the proxy server because for a new RTP session the client agent instantiates new objects and threads that increase the necessary CPU power. In general, the differences of CPU usage among the different streaming protocols is due to different software resources necessary on both proxies to manage the sessions.

Fig.7.d shows the battery consumption on the client and server nodes from 1 to 5 ongoing RTSP sessions with and without proxies. As yo can see, same values are obtained for all the experiments irrespective of the proxies usage or not. The battery consumption on the client node is higher than on the server node due to the CPU usage of the player to uncompress and play the stream.

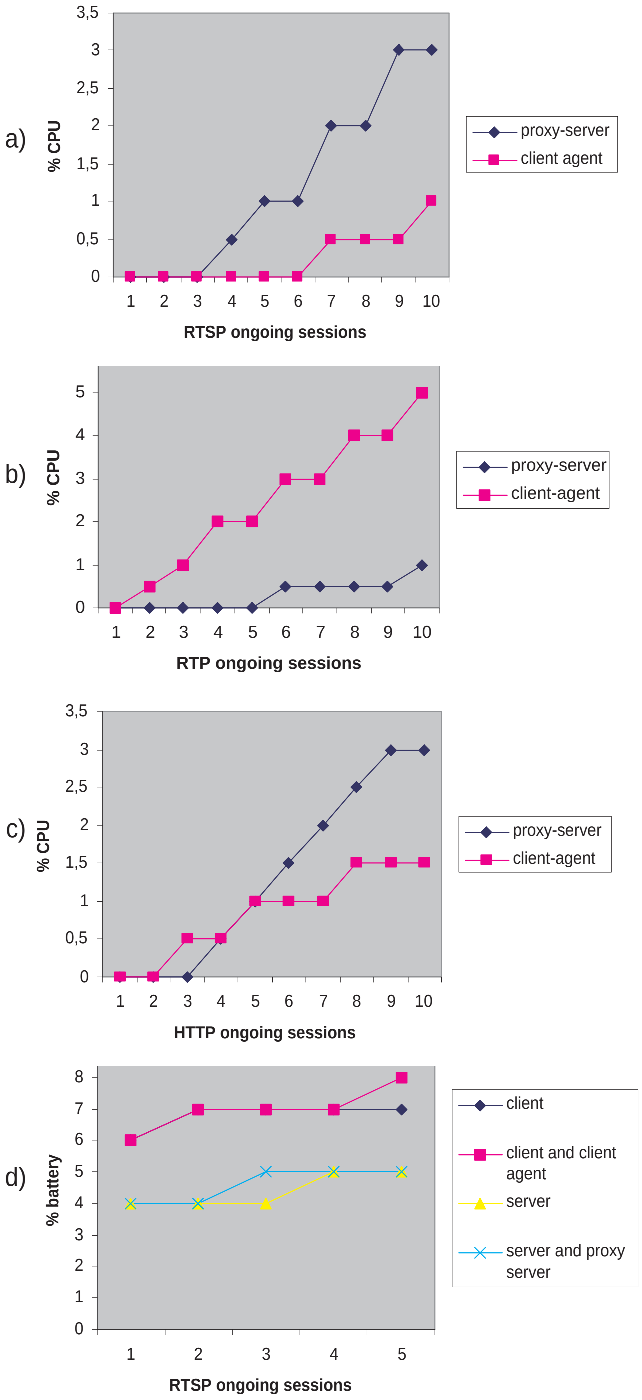

Figure 7: CPU consumption for: RTSP (a), RTP (b) and HTTP sessions (c). Battery consumption for RTSP sessions (d).

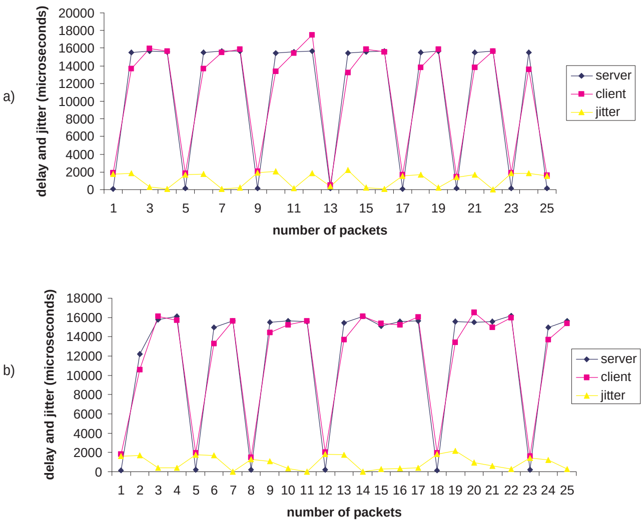

Figure 8: Delay and jitter for 25 samples. Without proxies (a), With proxies (b).

6.3 Delay and Jitter

Fig.8.a shows the delay in microseconds to transmit from 1 to 25 consecutive packets from the server to the client when only olsrd, the server and the player are running. It is also presented the jitter. The average jitter is 1.13608 ms, a viable value for the streaming transmission. Fig.8.b shows the same results when also the proxies are running. In this case, the average jitter is lower (0.94224 ms).

6.4 Corrective Actions

Fig.9.a presents the number of packets per second injected by the server during a VoD RTSP streaming session at a rate of 2000Kbps (green curve). The red curve is the OLSR traffic transmitted by the client node. We forced a disconnection period of 8s (about the 8^{th} second to the 16^{th} second). During this time interval, Ethereal tool did not capture OLSR traffic since the client is out of coverage but it captured RTSP traffic transmitted by the server since the server is not aware of the disconnection period (no proxies were used for this experiment). As a result, about 2MB are transmitted using RTP protocol and lost since we don't use a proxy server to pause the server. To correct this inefficient usage of the server and the available wireless bandwidth, we repeated the experiment using the proxies and we forced a higher disconnection period of 35s (Fig.9.b). During this period, the server is paused and no frames are transmitted

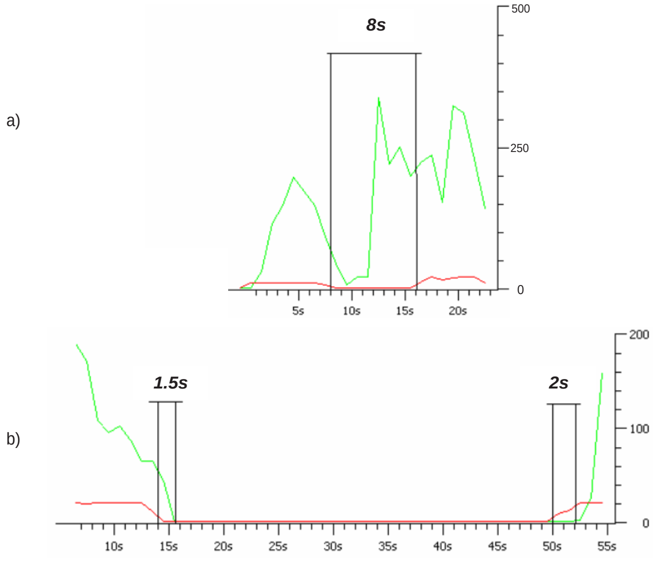

Figure 9: Bandwidth usage: Without proxies (a), With proxies (b).

avoiding that the server injects a total of 8.75 MB. As you can see, the proxy server lasts about 1.5s to detect the disconnection and about 2s to detect the reconnection. Both values are a bit higher to the theoretical value of 1 second we fix to warn the proxy about a disconnection or reconnection. Since we use a proactive protocol to detect them, the detection time is even lower that the one we would obtain using reactive protocols such as *Ad hoc On-demand Distance Vector Routing* (*AODV*) or *Dynamic Source Routing* (*DSR*) [24]. Similar results were obtained for RTP and HTTP sessions.

Fig.10 shows the behavior of the TCPControl mechanism for the RTSP session. Initially, the port used for the TCP connection between the client agent and the proxy server is 1273 (blue curve). About the second 17, the disconnection takes place and using TCPControl a new TCP connection over port 1280 is created (violet curve) and the streaming session is resumed and not abruptly ended after the reconnection. Similar results were obtained for HTTP sessions.

7 Conclusions and Future Work

This paper discusses some challenges that a video streaming session faces in a MANET. Path breaks between the client and the video streaming server lead to TCP connection failures, losing of frames and bandwidth and battery wastage. One of the important resources, the available bandwidth, must be used efficiently. For doing that, we proposed some corrective actions at the application layer for different streaming media protocols. Our solution is based on the usage of proxies to detect path breaks and reconnections thanks to the feedback provided by the proactive OLSR protocol. Another important effect because of the partitioning of the network is the TCP connection failure for HTTP and RTSP streaming sessions. It provokes an abort that could lead the session release. In this paper, we also proposed a solution for this problem. Experimental results showed the convenience of our mechanism to detect path breaks and reconnections and the benefits of using the corrective actions.

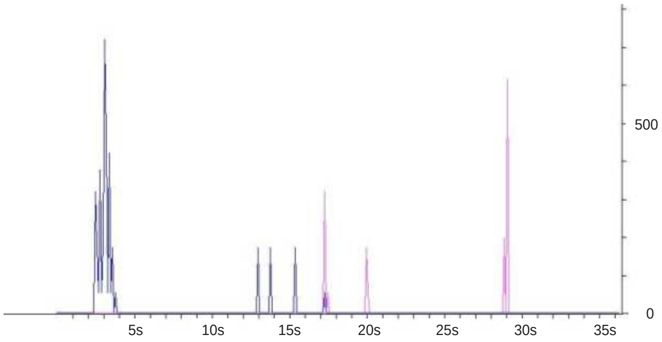

Figure 10: Management of long disconnections over TCP.

Some things remain to be done. For example, the enhancements to RTP and HTTP sessions for VoD in order to avoid losing of frames during the disconnections. Another additional corrective actions such as giving directions to the user to move to a better performance and coverage area would decrease the disconnection period. This is useful for live streaming sessions since the user does not lose too many frames, and also for VoD streaming sessions since the proxy server does not have to face the problem of storing frames during the disconnection for streams that can not be paused (e.g. streaming sessions over HTTP or RTP).

References

[1] *IEEE 802.11, The Working Group Setting the Standards for Wireless LANs*, http://grouper.ieee.org/groups/802/11/.

[2] Cunningham, G., Murphy, S., Murphy, L., Perry, P. *Seamless Handover of Streamed Video over UDP Between Wireless LANs*, Consumer

Communications and Networking Conference (CCNC), pp. 284–289, 2005.

[3] Schulzrinne, H., Casner, S., Frederick, R., Jacobson, V. *RTP: A Transport Protocol for Real-Time Applications*, http://www.ietf.org/rfc/rfc3550.txt, July 2003.

[4] Li, J., Li, L. *Research of Transmission and Control of Real-time MPEG-4VideoStreaming for Multi-channel Over Wireless QoS Mechanism*, 1^{th} International Multi-Symposiums on Computer and Computational Sciences, vol. 2, pp. 257–261, 2006.

[5] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T. *Hypertext Transfer Protocol – HTTP/1.1*, http://www.ietf.org/rfc/rfc2616.txt, June 1999.

[6] Kumar, K.G., Lipscomb, J.S., Ramchandra, A., Chang, S.S.P., Gaddy, W.L., Leung, R.H., Wood, S., Liang-Jie, Z., Chen, J., Menon, J.P. *The HotMedia Architecture: Progressive and Interactive Rich Media for the Internet*, IEEE Transactions on Multimedia, vol. 3, no. 2, pp. 253–267, 2001.

[7] Schulzrinne, H., Rao, A., Lanphier, R. *Real Time Streaming Protocol (RTSP)*, www.ietf.org/rfc/rfc2326.txt, April 1998.

[8] Schulzrinne, H., Rao, A., Lanphier, R., Westerlund, M., Narasimhan, A. *Real Time Streaming Protocol (RTSP)*, Internet Draft, Internet Engineering Task Force, February 2004, Work in progress.

[9] Fu, Z., Meng, X., Lu, S. *A Transport Protocol for Supporting Multimedia Streaming in Mobile Ad Hoc Networks*, IEEE Journal on Selected Areas in Communications, vol. 21, issue 10, pp. 1615–1626, 2003.

[10] Wei, Y., Bhandarkar, S.M., Chandra, S. *A Client-side Statistical Prediction Scheme for Energy Aware Multimedia Data Streaming*, IEEE Transactions on Multimedia, vol. 8, issue 4, pp. 866–874, 2006.

[11] Sunderam, V., Pascoe, J., Tonev, G. *Reconciling the Characteristics of Wired and Wireless Networks: The Janus Approach*, 4^{th} Annual International Workshop on Active Middleware Services, 2002.

[12] Macías, E.M., Suárez, A., Martín, J., Sunderam, V. *Using OLSR for Seamless Streaming Video in 802.11 Ad Hoc Networks*, International Multiconference of Engineers and Computer Scientists (IMECS), pp. 932–937, Kowloon, Hong Kong, 2006.

[13] Clausen, T.H., Jacquet, P., *Optimized Link State Routing Protocol, RFC3626, Internet Engineering Task Force (IETF)*, http://ietf.org/rfc/rfc3626.txt, October 2003.

[14] *OLSR.ORG*, http://www.olsr.org/.

[15] Chandran, K., Raghunathan, S., Venkatesan, S., Prakash, R. *A Feedback Based Scheme for Improving TCP Performance in Ad Hoc Wireless Networks*, International Conference on Distributed Computing Systems, pp. 472-479, 1997.

[16] Kopparty, S., Krishanmurthy, S.V., Faloutsos, M., Tripathi, S.K. *Split TCP for Mobile Ad Hoc Networks*, IEEE GLOBECOM, vol. 1, pp. 138-142, 2002.

[17] Holland, G., Vaidya, N. *Analysis of TCP Performance Over Mobile Ad Hoc Networks*, ACM MOBICOM, pp. 219-230, 1999.

[18] Kim, D., Toh, C.K., Choi, Y. *TCP-Bus: Improving TCP Performance in Wireless Ad Hoc Networks*, ICC, vol. 3, pp. 1707-1713, 2000.

[19] Liu, J., Singh, S. *ATCP: TCP for Mobile Ad Hoc Networks*, IEEE Journal on Selected Areas in Communications, vol. 19, no. 7, pp. 1300-1315, 2001.

[20] Liu, J., Singh, S. *ATP: Application Controlled Transport Protocol for Mobile Ad Hoc Networks*, IEEE WCNC, vol. 3, pp. 1318-1322, 1999.

[21] Sundaresan, K., Anantharaman, V., Hsieh, H.Y., Sivakumar, R. *ATP: A Reliable Transport Protocol for Ad Hoc Networks*, ACM MOBIHOC, pp. 64-75, 2003.

[22] *VideoLAN - Free Software and Open Source Video Streaming Solution for Every OS!*, http://www.videolan.org/.

[23] *Ethereal: A Network Protocol Analyzer*, www.ethereal.com.

[24] Siva Ram Murthy, C., Manoj, B.S. *Ad Hoc Wireless Networks. Architectures and Protocols*, Prentice Hall PTR, 2004.