ISSN: 2321-7782 (Online)
Volume 2, Issue 7, July 2014
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Secured Multimedia File Transfer over Wireless Network Using LZW Algorithm

B. Muzahadulla Khan[1]
Scholar, Department of CSE
MITS
Madanapalle – India

K. Madhava Kumar[2]
Scholar, Department of CSE
MITS
Madanapalle – India

M. Dinesh[3]
Scholar, Department of CSE
MITS
Madanapalle – India

Abstract: Now a day the broad casting of video plays a major role in wireless network. If we want to broadcast a video it takes more time to transmit from source to destination, due to large size of video. In wireless network the packet must be lightweight, so that it takes less amount of time to transmit Multimedia File. To achieve this we use Compression Technique for reducing the file size to make the packet light weight. Once the packet is light we should use the shortest path algorithm for transmitting the file from source to destination. For efficient transmission of packets we use multi path approach in Wireless sensor.

Keywords: Wireless Network, MANET, Video Stream, Video Broadcast, LZW

I. INTRODUCTION

Any Communication, Compression is useful because it helps to reduce the storage space. It means consumption of resources is reduced. Lossy compression means to loss some original data. Lossless compression is to get original data when decompressing. Data compression means to reduce the size of file. Audio compression means to reduce the bit rate to represent the Analog signal. Image compression is an important issue in Internet, mobile communications, digital library, digital photography, multimedia, teleconferencing applications. Image compression research aims at reducing the number of bits needed to represent an image by removing the spatial and spectral redundancies as much as possible. Compression is a process to decrease the quantity of data without extremely dipping the quality of the multimedia data. The transition and storing of compressed multimedia data is much quicker and more capable than original uncompressed multimedia data. There are several techniques and principles for multimedia data compression, particularly for image compression such as the JPEG and JPEG2000 standards. These principles consist of several functions such as color space conversion and entropy coding.

II. RELATED WORK

Audio compression can be estimated by Speed of Compression & decompression, Degree of Compression, Robustness and error correction, product support. When we compress audio, decoded signal should be as close as possible to original signal. It should have lowest implementation complexity. Algorithm for audio compression like mp3 is planned to greatly decrease the quantity of data necessary to signify the audio recording and still sound like a realistic reproduction of the original uncompressed audio for the majority viewers. For audio, human ear is the concluding judge of sound quality and it was performed as a "blind test". In video compression [4], it compresses video streams. Video streams like successive discrete images. Consecutive images are highly consistent. Barring cut shots or scene changes, any given video frame is probable to bear a close resemblance to adjacent frames. Video compression algorithm like MPEG exploits this strong correlation to accomplish far improved compression rates that would be possible with isolated images.

Selective Repeat Sequence

Selective Repeat is one of the automatic repeat-request (ARQ) techniques. With selective repeat the sender sends an amount of frames particular by a window size even with no need to stay for individual ACK from the receiver as in stop-and-wait. However, the receiver sends ACK for every frame independently, which is not like increasing ACK as used with go-back-n. The receiver accepts unfeasible frames and buffers them. The sender independently retransmits frames that have timed out. It may be used as procedure for the deliverance and acknowledgement of message units, or it may be used as procedure for the delivery of subdivided message subunits. When used as the procedure for the release of messages, the sending process continues to send a quantity of frames specified by a window size even after a frame loss. Unlike Go-Back-N ARQ, the receiving process will go on accept and acknowledge frames sent after a preliminary error this is the general case of the sliding window protocol with both broadcast and receive window sizes greater than 1. Receiver method keeps track of the series number of the initial frame it has not received, and ends that quantity with every acknowledgement (ACK) it sends. If a frame from the sender does not arrive at your destination from the receiver, the sender continues to send ensuing packet until it has empty its window. The receiver continues to fill its receiving window with the consequential frames, reply every time with an ACK containing the series number of the earliest missing frame. Once the sender has sent all the frames in its window, it resends the frame numeral given by the ACKs, and then continues where it left off. The size of the sender and receiver windows must be indistinguishable, and half the maximum series number (presumptuous that series numbers are numbered from 0 to (n−1)) to avoid miscommunication in all cases of packets being dropped. To know this, consider the case when all ACKs are destroyed. If the receiving window is bigger than half the maximum series number, some perhaps even all, of the packages that are resent after timeouts are duplicates that are not predictable as such. The sender moves its window for every packet that is acknowledged.

III. PROPOSED WORK

In this paper, we use routing information for the following reasons:

This Compression technique is used to decrease the amount of data without enormously reducing the quality of the multimedia data. The evolution and storing of compressed multimedia data is much more rapidly and more efficient than innovative uncompressed multimedia data.

1.1 COMPRESSION TECHNIQUE

- LZW Encoding Algorithm

If the message to be encoded consists of only one character, LZW outputs the code for this character otherwise it inserts two or multicharacter, overlapping, different patterns of the message to be encoded in a Dictionary. The last character of a prototype is the first character of the next pattern.

The patterns are of the form: C0 C1. . . Cn-1 Cn . The prefix of a prototype consists of all the pattern characters except the last: C0 C1 . . . Cn-1

LZW output if the message consists of more than one character:

- If the pattern is not the last one output: The code for its prefix.
- If the pattern is the last one:
    - ➢ If the last pattern exists in the Dictionary; output: The code for the pattern.
    - ➢ If the last pattern does not exist in the Dictionary; output: code (lastPrefix) then output: code (lastCharacter)

LZW Encoding Algorithm

Initialize Dictionary with 256 single character strings and their corresponding ASCII codes;

```
Prefix ← first input character;
Codeword ← 256;
While (not end of character stream){
Char ← next input character;
If (Prefix + Char exist in the Dictionary)
Prefix ← Prefix + Char;
else {
Output: the code for Prefix;
InsertInDictionary ( (Codeword , Prefix + Char) ) ;
Codeword++;
Prefix ← Char;
}
}
```

LZW Decoding Algorithm

The LZW decompressor creates the same string table throughout decompression. Initialize Dictionary with 256 ASCII codes and analogous single character strings as their translations.

```
Previous Codeword ← first input code;
Output: string (Previous Code word);
Char ← character (first input code);
Codeword ← 256;
While (not end of code stream)
{
Current Codeword ← next input code;
If (Current Codeword exists in the Dictionary)
String ← string (Current Codeword);
else
String ← string (Previous Codeword) + Char;
Output: String;
Char ← first character of String;
Insert in Dictionary ((Codeword, string (Previous Codeword) + Char));
Previous Codeword ← Current Codeword;
Codeword++
Output: string (first Codeword);
```

```
While (there are more Codeword){
If (Current Codeword is in the Dictionary)
Output: string (Current Codeword);
else
Output:
Previous Output + Previous Output first character;
insert in the Dictionary:
Previous Output + Current Output first character;
}
```

1.2 SHORTEST PATH ROUTING

Kruskal's is the most efficient shortest path routing algorithm for transmission of multimedia file in wireless networks. This approach has better performance when edges are high in a network. And also it considers only the edges in directed acyclic graph for transmission of file.

The algorithm maintains a forest of trees, an edge is accepted if it connects vertices of distinct Trees. We need a data structure that maintains a partition, i.e., a collection of disjoint sets, with the following operations.

- find(u): return the set storing u

- union(u,v): replace the sets storing u and v with

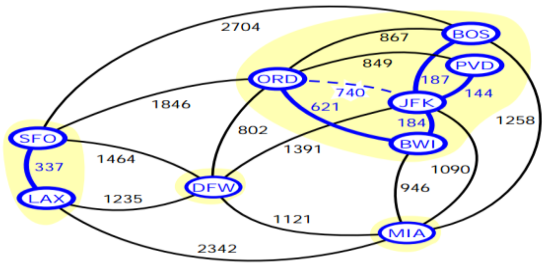

Figure 1 : graph

Representation of a Partition

Each set is stored in a sequence

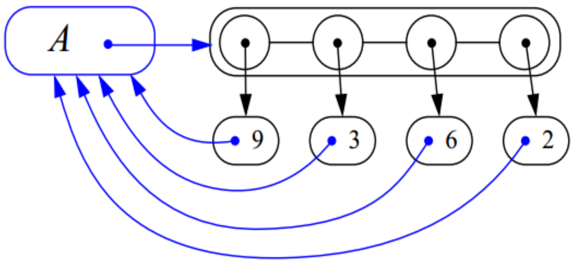

Figure 2: Representation of partition

• Each element has a reference back to the set operation find(u) takes O(1) time in operation union(u,v), we move the elements of the smaller set to the sequence of the larger set and update their references

• The time for operation union (u,v) is min(nu , nv),where nu and nv are the sizes of the sets storing u and v . Whenever an element is processed, it goes into a set of size at least double. Hence, each element is processed at most log n times.

Pseudo Algorithm Kruskal(G):

```
Input: A weighted graph G.
Output: A minimum spanning tree T for G.
Let P be a partition of the vertices of G, where each vertex forms a separate set
Let Q be a priority queue storing the edges of G and their weights
T ← Ø
while Q ≠ Ø do
(u,v) ← Q.removeMinElement ()
if P.find(u) ≠ P.find(u) then
add edge (u,v) to T
P.union (u,v)
return T
Running time: O ((n+m) log n)
```

Analysis

Compare Prim and Kruskal

• Both have the same output → MST

• Kruskal's begins with forest and merge into a tree

• Prim's always stays as a tree

•If you don't know all the weight on edges → use Prim's algorithm

• If you only need partial solution on the graph → use Prim's algorithm

Complexity

Kruskal: O (NlogN)

Comparison sort for edges

Prim: O (NlogN)

Search the least weight edge for every vertex

Analysis of Kruskal's Algorithm

Running Time = O (m log n) (m = edges, n = nodes)

Testing if an edge creates a cycle can be slow unless a complicated data structure called a "union-find" structure is used. It usually only has to check a small fraction of the edges, but in some cases (like if there was a vertex connected to the graph by only one edge and it was the longest edge) it would have to check all the edges. This algorithm works best, of course, if the number of edges is kept to a minimum.

Analysis of Prim's Algorithm

Running Time = O (m + n log n) (m = edges, n = nodes)

If a heap is not used, the run time will be O(n^2) instead of O(m + n log n).

However, using a heap complicates the code since you're complicating the data structure. A Fibonacci heap is the best kind of heap to use, but again, it complicates the code. Unlike Kruskal's, it doesn't need to see the entire graph at once. It can deal with it one piece at a time. It also doesn't need to worry if adding an edge will create a cycle since this algorithm deals primarily with the nodes, and not the edges. For this algorithm the number of nodes needs to be kept to a minimum in addition to the number of edges. For small graphs, the edges matter more, while for large graphs the number of nodes matters more. Kruskal's has better running times if the number of edges is high, while Prim's has a better running time if both the number of edges and the number of nodes are low.

Go-Back-N ARQ is a specific instance of the automatic repeat request (ARQ) protocol, in which the distribution process continues to send a number of frames specified by a window size even without receiving an acknowledgement (ACK) packet from the receiver. It is scrupulous case of the general sliding window protocol with the transmit window size of N and receive window size of 1.The receiver procedure keeps track of the series number of the next frame it expects to receive, and sends that number with every ACK it sends.The receiver will ignore any frame that does not have the exact series number it expects – whether that frame is a "past" duplicate of a frame it has already ACK'ed [1] or whether that frame is a "prospect" frame past the last packet it is waiting for. Once the sender has sent all of the frames in its *window*, it will distinguish that all of the frames since the first mislaid frame are outstanding, and will go back to sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again.

Go-Back-N ARQ is an additional capable use of a link than Stop-and-wait ARQ, since different waiting for an acknowledgement for each packet the link is still being utilized as packets are being sent. In other words, during the time that would or else be spent waiting, more packets are being sent. However, this process also consequences in sending frames multiple times – if any frame was lost or damage, or the ACK acknowledging them was gone or damaged, then the frame and all follow frames in the window (even if they were received without error) will be re-sent. To stay away from this, Selective Repeat ARQ can be used.

N = window size

Rn = request number

Sn = sequence number

Sb = sequence base

Sm = sequence max

Receiver:

Rn = 0

Do the following forever:

If the packet received = Rn && the packet is error free

Accept the packet and send it to an upper layer

Rn = Rn + 1

Send a Request for Rn

Else

Refuse packet

Send a Request for Rn

Sender:

Sb = 0

Sm = N − 1

Repeat the following steps forever:

1. If you receive a request number where Rn > Sb

Sm = Sm + (Rn − Sb)

Sb = Rn

2. If no packet is in transmission,

Transmit a packet where Sb <= Sn <= Sm.

Packets can transmit in order.

IV. Conclusion

This Lzw Compression technique is used to decrease the quantity of data without enormously reducing the quality of the multimedia data. The evolution and storing of compressed multimedia data is much faster and well-organized than original uncompressed multimedia data. Finally we show the efficient way of transmitting the media file from source to destination by using the following approaches. As per analysis the kruskal's approach had better performance than prim's where nodes and edges are consider in broadcasting multimedia files in large wireless network. The select repeat sequence approach helps in faster distributing and merging the media file packets at both source and destination ends. The goal of this project is to explore energy-efficient protocols in broadcasting scenarios and compare a suitable protocol with three other broadcast protocols in Wireless network. We adopted the multipoint relay selection strategy based on residual energy in the EOLSR protocol and use it in the broadcasting scenarios. This EMPR selection strategy takes into account the energy dissipated in transmission and reception up to 1-hop from the transmitter and was verified to prolong the network lifetime and increase the packet delivery rate when combined with the proposed unicast routing strategy. The select repeat sequence approach helps in faster distributing and merging the media file packets at both source and destination ends. The future work in the design of energy-efficient broadcast routing protocols in wireless should try to reduce the transmission redundancy and overall network overhead, and thus achieve the minimum energy consumption and the maximum network lifetime.

References

1. M.Dinesh, E.Madhusudhana Reddy, Ultimate Video Spreading With Qos over Wireless Network Using Selective Repeat Algorithm. International Journal of Computer Science Engineering (IJCSE) Volume 2 No.04 July 2013, pp 96-102.
2. D. Johnson, Y. Hu, D. Maltz, Dynamic Source Routing Protocol for Mobile Ad Hoc Networks for IPv4, RFC 4728
3. C. Perkins, E. Belding-Royer, S. Das, Ad hoc On-demand Distance Vector (AODV) Routing, RFC 3561
4. C. E. Perkins, P. Bhagwat, Highly Dynamic Destination-Sequenced Distance Vector (DSDV) for Mobile Computers, Proc. of the Special Interest Group on Data Communication, Volume 24 , Issue 4, Oct. 1994, pp. 234-244.
5. P. Jacquet, T. Clausen, Optimized Link State Routing Protocol (OLSR), RFC 3626
6. S. Lee, M. Gerla, C. Chiang, On-Demand Multicast Routing Protocol, Proc. of IEEE Wireless Communications and Networking Conference, Sep. 1999, pp. 1298-1304.
7. E. M. Royer, C. E. Perkins, Multicast Operation of the Ad-hoc On-demand Distance Vector Routing Protocol, Proc. of ACM/IEEE International Conference on Mobile Computing and Networking, Aug. 1999, pp. 207-218.
8. E. Bommaiah, A. McAuley, R. R. Talpade, M. Liu, AMRoute: Ad hoc Multicast Routing Protocol, Internet-Draft, draft-talpade-manetamroute-00.txt, Work in progress, Aug. 1998.
9. Network Working Group, SMF Design Team (IETF MANET WG), Simplified Multicast Forwarding for MANET (draft-ietf-manet-smf-07), Feb. 2008.
10. L. Li, R. Ramjee, M. Buddhikot, S. Miller, Network Coding-Based Broadcast in Mobile Ad hoc Networks, 26th IEEE International Conference on Computer Communications, IEEE, May 2007, pp. 1739-1747.

11. S. Mahfoudh, P. Minet, EOLSR: An EnergyEfficient Routing Based on OLSR in Wireless Ad Hoc and Sensor Networks, Proc. of the 22nd International Conference on Advanced Information Networking and Applications, Mar. 2008,pp.1253-1259**.**

12. Efficient Video Broadcasting over Wireless Network through EOLSR, International Journal of Advanced Research in Computer Science and ApplicationsVol.1, Issue 1, July 2013,pp 34-41.

13. P. Jacquet, T. Clausen, Optimized Link State Routing Protocol (OLSR), RFC 3626

AUTHOR(S) PROFILE

B. Muzahadulla Khan received his M.Tech degree in Computer Science & Engineering from Madanapalle Institute of Technology & Science, Madanapalle, Angallu, 517325, Chittoor (DT), Andhra Pradesh, India in 2013 and B.Tech in Computer Science & Engineering from Kuppam Engineering College, Kuppam, Andhra Pradesh, India in 2010.

K. Madhava Kumar received his M.Tech degree in Computer Science & Engineering from Madanapalle Institute of Technology & Science, Madanapalle, Angallu, 517325, Chittoor (DT), Andhra Pradesh, India in 2013 and B.Tech in Information Technology from Alfa College of Engineering & Technology, Allagadda, Andhra Pradesh, India in 2010.

M. Dinesh received his M.Tech degree in Computer Science & Engineering from Madanapalle Institute of Technology & Science, Madanapalle, Angallu, 517325, Chittoor (DT), Andhra Pradesh, India in 2013 and B.Tech in Computer Science & Engineering from Vaagdevi Institute of Technology & Science, Proddatur, 516360, YSR(Dist), Andhra Pradesh, India in 2010.