Faculty ECI

Abdul Jabbar

Analyzing Online Game Traffic in Hybrid Mobile Ad Hoc Wireless Networks with/without Packet Aggregation

Computer Science
D-level thesis

Date/Term:	08-06-04
Supervisor:	Andreas J. Kassler
Examiner:	Donald F. Ross
Serial Number:	D-20 08 : 04

Karlstads universitet 651 88 Karlstad
Tfn 054-700 10 00 Fax 054-700 14 60
Information@kau.se www.kau.se

Analyzing Online Game Traffic in Hybrid Mobile Ad Hoc Wireless Networks with/without Packet Aggregation

Abdul Jabbar

This report is submitted in partial fulfillment of the requirements for the Master's degree in Computer Science. All material in this report which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Abdul Jabbar

Approved, 08-06-30

Advisor: Andreas J. Kassler

Examiner: Donald F. Ross

Abstract

Today's internet traffic significantly consists of network game traffic. This type of traffic is interesting with respect to its market potential and real-time requirements over network. This kind of traffic has completely different requirements as compared to the classical services therefore some optimizations are required to improve performance. In this thesis study, Quake3 game is studied in a MANET, which is a highly interactive online first person shooter game. These types of games require a minimum quality of service from network since they are fast paced and require quick user response. MANET is a network consisting of two or more mobile nodes connected wirelessly without any network centralized infrastructure. It is connected to external internet by the help of mobile nodes that act as gateways. Such a network is known as hybrid MANET. Providing better quality of service for games like Quake3 over MANET is quite challenging that requires some optimizations. In this paper, Quake3 game traffic is aggregated and then sent over internet connected wireless network. The basic goal of this thesis is to evaluate the performance of Quake3 over hybrid MANET with packet aggregation and no-aggregation. This is achieved by performing simulations in Network Simulator 2 (2.26). The results obtained suggest that packet aggregation showed better results for improving game performance in hybrid MANET.

Contents

List of Figures

List of tables

List of Equations

List of Codes

List of Graphs

1 Introduction

1.1 Background

Online games, in the past few years, have emerged as quite popular, big and profit promising business in today's world of internet. Internet traffic is composed of significant game traffic. According to [1] about 4% of all traffic running through the backbone could be because of only 6 popular games, just only in USA, from a fraction of 7 billion computer games industry [1]. Most of the interest of game users is focused on fast moving, quick paced and highly interactive "First Person Shooter" (FPS) games. Reason behind calling them as First Person Shooter is that the game world is rendered visually with respect to the player character, where the main character is continuously on the move, running and picking up weapons and ammunition. Only hand with guns can be seen of the playing player so that more concentration can be put towards shooting. In FPS games, any player's survival greatly depends on its fast movement, quick decision and right aim. Otherwise will be spot by any other player and killed. Therefore, for the game players, it is very much critical that game maps and players movements must be updated as fast as possible, otherwise the game will be non playable. The game designers have designed these games to be played on LAN or internet. In order to play these games in wireless environment, several changes to the infrastructure are required since, unlike internet, there are no fixed connections between nodes and high probability of packet drops and weak signals resulting in time varying delays.

As the time is passing, devices are constantly getting increasingly powerful according to the needs and requirements of their users. Devices like mobile phones, laptops and PDA's are now equipped with wireless networking. This has resulted in shifting the trend of playing games from internet to wireless networks. A wireless network can be composed of an infrastructure like base stations and access points or without any infrastructure i.e. mobile ad-hoc wireless network (MANET).

MANET is a wireless network in which the mobile nodes communicate with each other directly and independent of any infrastructure of base stations or access points. There is no central controlling authority which makes the nodes self-organizing and self-managing. Because of these properties, MANETs are highly flexible, self organize able and easily and quickly deployable. Network topology is dynamically changing since most or all of the nodes are mobile. Any node in the network can be a router or a host at the same time.

Communication between nodes can be from multiple hops. Since the nodes are moving, therefore they are operating on batteries resulting in an energy constraint requiring power efficient operations. Also the wireless nodes must manage the routing and host information by their-self since there are no base stations, making routing more complex in wireless ad-hoc networks [7].

One can think that what the basic need behind MANET network was, like, why there was a need to use such an infrastructure-less network having quite limitations despite of the presence of more systematic and infrastructure based networks. Answer to this question is that there are such situations in which a running functional network is required as quick as possible independent of any existing infrastructure, e.g. military or any emergency situation where independency and time are basic factors. Later, MANETs became commercial and started using in miscellaneous applications such as group travelling, location-specific services, home network, vehicle network, emergence response, search and rescue, disaster relief, law enforcement, military, surveillance network and multiplayer gaming.

Online games are becoming interestingly popular day by day in MANETs since they can be played among co-located people that are part of the wireless network, without the need of any underlying infrastructure anywhere, anytime. They don't have any kind of restriction to be present at a special place on specific time.

FPS game traffic falls into the category of real-time traffic which has strict requirements like small delay, low packet loss rate and jitter, fast packet delivery ratio. If we see at the conditions of an FPS game, played in a given network, delay must be in between 50-150 ms, having minimum jitter and packet loss ratio below 5% [2]. In MANETs, fulfilling these requirements is a challenging job since they are not designed for real-time applications.

In order to increase performance of online games being played in MANETs, I have applied the idea of packet aggregation proposed in [3] to the traffic of an FPS game Quake3 and obtained remarkable results.

1.2 Proposed Goals

Following are the proposed goals of this master's thesis:

1. To give an overview of internet connected MANETs.
2. Overview of routing protocol AODV-UU [13].
3. Explain the idea of packet aggregation as proposed in [3].
4. Introduction to NS2 [5].

5. Explain the game traffic model of Quake3 as proposed in [4].
6. Applying packet aggregation to game traffic.
7. Performing simulations in network simulator ns-2.
8. Obtaining results about delay, jitter and packet loss as compared to game traffic with aggregation and without aggregation.

1.3 Document Outline

Structure of this thesis is as follows:

- Chapter 2 contains the background of internet connected MANETs, AODV-UU, packet aggregation, quake3 game traffic model and network simulator NS2.
- Chapter 3 contains design and implementation of simulations with packet aggregation and without aggregation. This section contains sample code from simulations and explains its working.
- Chapter 4 explains the general simulation setup and basic scenario. It also describes individual simulations with nodes placement. Performance is also evaluated in this chapter for delay, jitter and packet loss. Results are evaluated and shown graphically with respect to packet aggregation and no-aggregation.
- Chapter 5, in the end, concludes this thesis study.

2 Background

This chapter puts light on the background information necessary about internet connected MANETs, routing protocol used, concept of packet aggregation and the proposed game model.

2.1 Internet Connected MANETs

Basically MANETs were designed to cope with situations where traditional networks were hard to implement like in emergency situations, whether it is a catastrophic or military. Studies in the past recent years have changed this trend and have shifted towards heterogeneous networks e.g. interconnecting mobile ad-hoc network with cellular network, wireless LAN or wired network like internet. This type of MANET is known as hybrid MANET. Internet connected MANETs have the advantages of fixed stations from fixed network and the multi-hop characteristics of mobile wireless networks. These types of networks have increased flexibility i.e. any mobile node having appropriate characteristics can be selected as gateway and reliability i.e. greater coverage because of multi-hop feature, useful for the areas having transmission problems. The idea for integrating MANETs with IP networks has become quite popular among the research community of late [6]. This combination has the potential to extend the ordinary internet access from its current scope to remote inaccessible areas resulting in availability of World Wide Web services anywhere anytime [9]. Hybrid MANET like ordinary MANET can also be deployed in a fraction of time with ease. Its applications are wireless classes, conference meetings, in trains, airports etc.

MANETs connect to internet by the help of multi-homed nodes which work as the gateways between MANET and external network. All the other nodes in MANET send their traffic to external network with the help of the gateway nodes i.e. traffic from the sender node is forwarded hop by hop from intermediate nodes till it reaches the gateway, from where it is forwarded to the external network. Same mechanism is employed for incoming traffic i.e. traffic from external network reaches at the gateway from where it is forwarded to the destination node by the help of packet forwarding from intermediate nodes. These gateways are responsible for understanding and translating protocols from wired network to the wireless network and vice versa.

2.1.1 Gateway Discovery

In order to have an internet access for the MANET, special type of intermediate nodes called gateways forward traffic to and from mobile wireless network and internet. Before starting of any type of communication, member nodes must locate the gateway node(s) of the MANET and select one of them. Discovering and selecting internet gateways highly affects the performance of hybrid ad-hoc wireless networks. A wireless node must choose the most appropriate gateway for accessing internet.

According to [8], gateway discovery can be defined as the capability of MANET to access internet or conventional network and provide global addressing and bidirectional reach-ability to MANET nodes [8].

Gateway discovery can be initiated by any mobile node. There are three types of approaches employed for gateway discovery.

2.1.1.1 Proactive Gateway Discovery Approach

In proactive approach [10], gateway discovery is initiated by the gateway itself i.e. it identifies itself as a gateway. Special types of messages called “Gateway Advertisements” are broadcasted periodically by the gateway to the mobile nodes present in its transmission range. Special care must be taken for determining the advertisement interval, otherwise the network might get flooded.

On receiving this advertisement, the mobile nodes create a route entry, if it is a new route, otherwise it updates its routing table for that gateway and then rebroadcasts the message to its neighboring mobile nodes. The resulting overhead is quite high since the network now contains lots of unnecessary duplicate gateway advertisement packets for the same gateway which are received by mobile nodes. Solution to this problem is adding an ID field to the broadcast message, containing the gateway ID. When duplicate gateway advertisement messages of same ID are received by a node, it just considers one and takes action respectively [11] and drops the duplicate gateway advertisement.

Advantage of this approach is that mobile node has the chance of initiating handover before losing its internet connection but the disadvantage is that gateway discovery advertisements are flooded periodically through whole MANET, resulting in a high overhead for mobile nodes having limited resources like bandwidth and energy (power) which will be utilized a lot.

2.1.1.2 Reactive Gateway Discovery Approach

In reactive approach, the gateway discovery is initiated by the mobile node that wants to create or update a route to the gateway. It broadcasts a special kind of route request packet known as RREQ_I [11] where “I” denotes the IP address for that group’s gateways. Since this message is only destined for gateways, therefore it is only processed by them. Intermediate mobile nodes on receiving RREQ_I just rebroadcast it. When the gateway receives this message, it unicasts back route reply called RREP_I containing the IP address of gateway.

The source node receives this RREP_I packet and chooses the gateway based on hop-counts, if there are more than one gateways. Here again we have duplicate packets because of forwarding of gateway discovery packets, which are handled in the same manner as in proactive case.

Advantage of this approach is that the network does not contain unnecessary control messages, they are generated only when a node wants to know about reachable gateway(s). This prevents from periodic flooding as in proactive approach. As disadvantage, when a node wants to know about the gateway, it will take longer time for determining the route. Also a mobile node loses its internet connection without initiating a handover.

2.1.1.3 Hybrid Gateway Discovery Approach

Hybrid gateway discovery approach is formed by the combination of proactive and reactive approaches in such a way that their drawbacks are minimized. This approach makes use of both approaches in such a way that mobile nodes present near the gateway use the proactive approach whereas the mobile nodes beyond these nodes employ the reactive approach for the gateway discovery.

For the mobile nodes near gateway, the gateway broadcasts gateway advertisement periodically. Mobile nodes after receiving this message, update their routing tables and then rebroadcast this message. Mobile nodes forward this gateway advertisement to a certain amount of hops that is determined by the advertisement zone. In other words, this is the range in which proactive approach is used.

For mobile nodes residing beyond this area, they will broadcast RREQ_I packet when they want to determine their gateways. This packet is rebroadcasted by the intermediate mobile nodes until it is received by a mobile node in the area near gateway which has the route information for gateway through proactive approach. Route reply packet RREP_I is unicasted from this mobile node to the source node [12].

2.2 AODV-UU

AODV-UU [13] (Ad-Hoc On-demand Distance Vector – Uppsala University) is an extended version of AODV [14] routing protocol developed at Uppsala University in Sweden having all the mandatory as well as some optional features of AODV containing some extra functionality i.e. gateway discovery and half tunneling. Like AODV, AODV-UU also falls into the category of reactive routing protocols.

Since AODV-UU is the extension of AODV therefore route discovery and route maintenance is same in both of them. I will explain the additional functionality of AODV-UU like gateway discovery and half tunneling.

2.2.1 Gateway Discovery

Since in MANETs, member mobile nodes rely only on the gateways for accessing internet therefore discovering and choosing a gateway is very important. Performance of a MANET is highly dependent on choosing right gateways and method employed for discovering gateways.

As mentioned earlier, AODV-UU is a reactive routing protocol, therefore gateway discovery mechanism is initiated by the mobile node that wants to send traffic to the outer network or internet. The source mobile node first broadcasts a route request packet which is forwarded by the intermediate nodes until it is reached at the gateway. In response, the gateways send back a unicast route reply packet. If there are multiple gateways present in the MANET from which the source node has received replies, then it chooses the gateway depending on hop counts [15].

After a route has been established, route maintenance procedure maintains it until the destination gets disconnected from every path or the route is not needed anymore and it times-out without refresh.

2.2.2 Half Tunneling

The concept of tunneling implies that a tunnel is established between the source and destination nodes. By doing so, the intermediate nodes need not to worry about maintaining the routing information. Communication in this manner appears as a one hop connection although, in actual, it is a multi-hop connection. Reason behind this is that, tunneling creates a virtual one hop path between source and destination nodes.

AODV-UU also uses the concept of tunneling but rather in a different fashion i.e. instead of bi-directional tunneling it uses half-tunneling [13]. According to this concept, a tunnel is

established on the single side of connection i.e. from the source node to the gateway. Traffic, from the wired network to MANET, first reaches at the gateway from where it is sent to its appropriate node since the gateway has the routing information about the wireless nodes or in other case it can obtain using normal route discovery. Intermediate wireless nodes of MANET do not need to worry about tunneling, they just have to forward these incoming packets as normal packets in wireless network. Just source node and gateway know that the packet is encapsulated and is for external network. Half tunneling reduces recursive route requests since intermediate nodes just need to forward these packets. This concept helps to reduce the routing table entries for intermediate wireless nodes since they just have to forward the packets to gateway whose route information, if available, is already present in their routing table.

Hence, tunneling creates a virtual one hop distance between source node and destination node. Half-tunneling of AODV-UU is more robust than default route in mobile ad-hoc wireless networks because a default route points to only one gateway at a time whereas in tunneling an intermediate hop can be shared by two nodes and tunnels to different gateways are also maintained at the same time.

2.2.3 Encapsulation

AODV-UU uses encapsulation in which the packet is encapsulated with a new header that contains the IP address of the gateway as the destination address, if the destination node is not in the MANET. After reaching at gateway, this packet is then de-capsulated and the original packet is then sent to the actual destination address. In this manner only the source node and destination node knows that encapsulated packets are sent.

For the incoming traffic for MANET, the packets are not encapsulated at the gateway. They are simply forwarded to the destination node in MANET since gateway has the address of nodes present in MANET.

2.3 Packet Aggregation

Wireless network data transmission rate as compared to the wired network is very low since it is less reliable and prone to low signal strength, disconnection, interference etc. As a result of this, IEEE 802.11 standard specifies an additional checksum for physical layer encapsulation along with MAC header and frame body. After a successful transmission, an acknowledgement is also sent by the receiver. Small packets are responsible for occupying most of the network resources since physical layer and MAC layer overhead is same for every

packet. According to [16], MAC is inefficient because of its overhead like headers, back-off time, inter-frame spaces, acknowledgements etc. It gets worse with the increase in data rate [17]. Several techniques are employed to cope with these limitations. One of them is “Packet Aggregation”. According to this technique, several different small packets are added together to create a new larger packet that is then transmitted. Packet aggregation can be applied on several different levels i.e. MAC-level [16] [17], IP-level [19] [20]. When smaller packets are aggregated, lots of overhead is reduced. When the packets are aggregated to form a new bigger packet, an additional header is added to it by the aggregator node. This header is used by the target node to de-aggregate bigger packets into respective smaller ones.

Packet aggregation can be seen in the following picture.

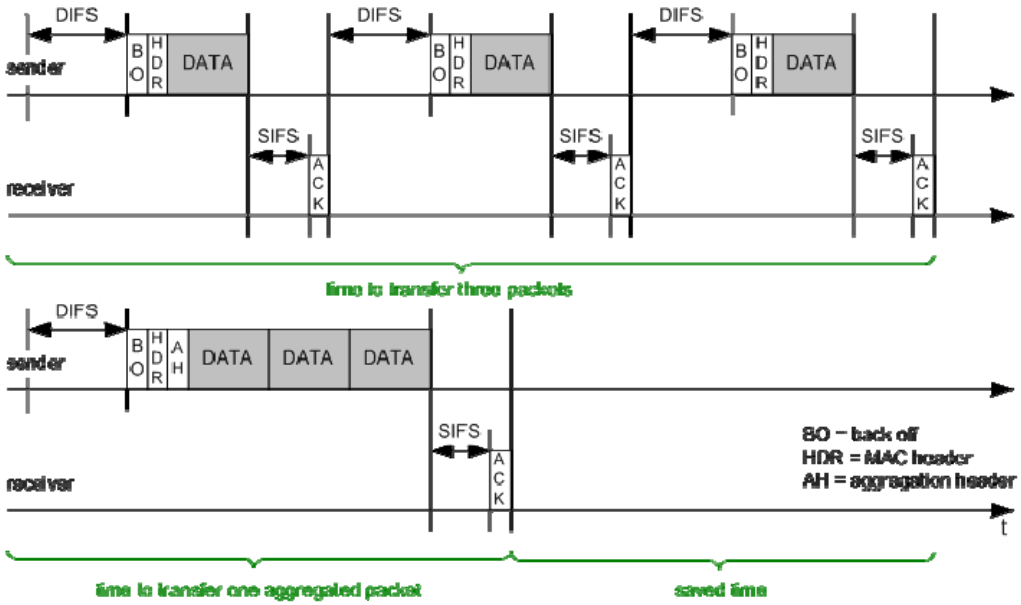


Figure 1: Packet aggregation from [45]

The above figure shows the difference between transferring three packets without and with packet aggregation. One aggregated packet is formed after combining three packets. When these packets are aggregated, an aggregation header is added by the sender node to the aggregated packet which is used by the de-aggregator node to de-aggregate them correctly. It is also seen in the above picture that packet aggregation reduces MAC and physical layer overhead and saves significant transmission time.

2.3.1 Classification of Packet Aggregation

Packet Aggregation can be classified into two categories i.e. end-to-and and hop-by-hop [18] [45] [46].

2.3.1.1 End-to-End Packet Aggregation

In end-to-end packet aggregation, all packets destined to a common node (destination) are aggregated. Packet aggregation is done by the sender (source) node with respect to the destination node. Additional delay can be added by the sender node to aggregate packets. Intermediate nodes do not process these aggregated packets. After receiving they just forward them to the next node. Aggregation is done once, by the source node, only. Destination node, after receiving these aggregated packets, de-aggregates them accordingly.

End-to-end aggregation can be seen in the following picture.

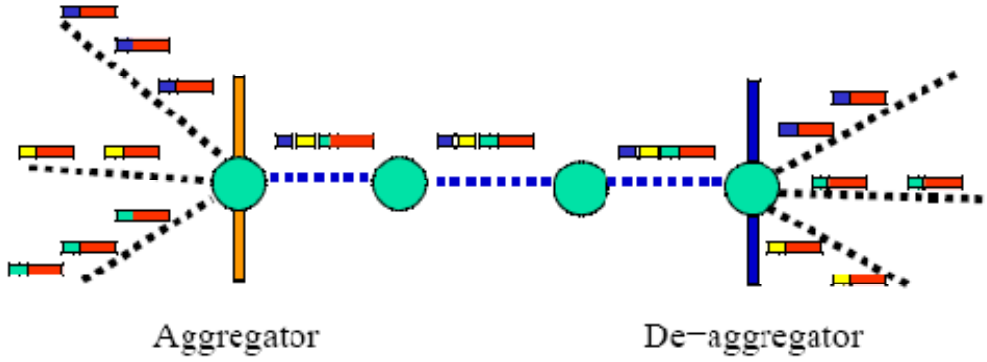


Figure 2: End-to-End Aggregation [47]

In the above figure, packets are aggregated from different VOIP calls into larger one at source node and then they are transmitted. Intermediate nodes do not need to process them. They are just forwarded to next nodes until they reach their destination. At de-aggregator node, they are de-aggregated into respective smaller packets.

2.3.1.2 Hop-by-Hop Packet Aggregation

In hop-by-hop packet aggregation, packets are aggregated and de-aggregated at every node. Forced delay can be added by the nodes at every hop. Packets are aggregated by the sender node first and then they are transmitted to next hop. After receiving these packets, this node first de-aggregates the packets, and then again aggregates these and other packets according to the next hop. Since packets are transmitted from one node to other node hop-by-hop, therefore additional delay is added at every hop resulting in cumulative increased delay. This scheme is rather complex since every node has to aggregate and de-aggregate packets.

Hop-by-hop packet aggregation can be seen in the following picture.

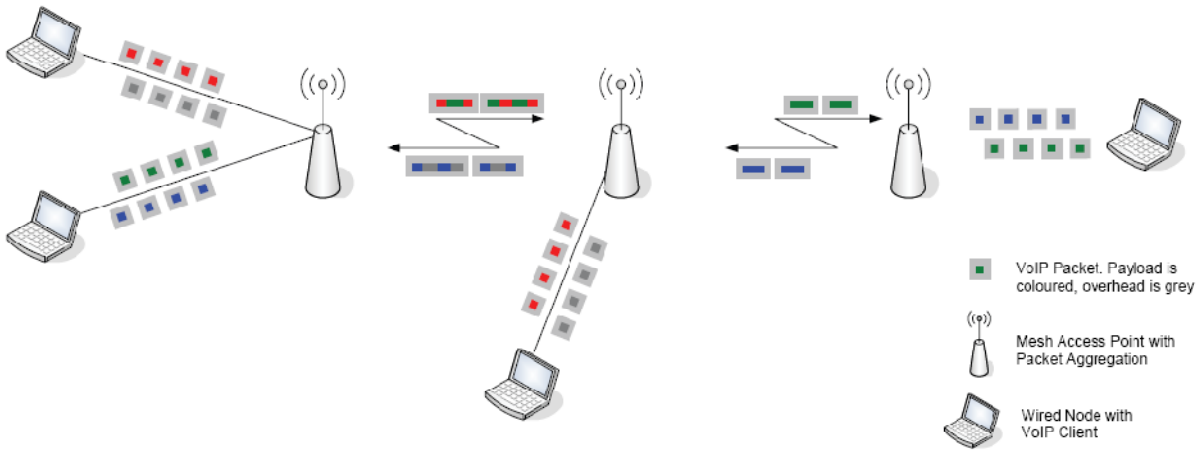


Figure 3: Hop-by-hop packet aggregation [3]

2.3.2 Simple Packet Aggregation Approach

Following is the simplest approach for packet aggregation as described in [18].

2.3.2.1 Forced-Delay Aggregation

According to this approach, all incoming packets are marked with some timestamp which corresponds to the maximum delay period of that particular packet. When this timestamp expires, all packets having same hop are aggregated. Maximum Transmission Unit (MTU) size limits the number of packets to be aggregated into larger packet. If there is some packet that cannot be aggregated, it is sent after its timestamp expires. Maximum delay period directly affects overall performance i.e. delay and packet size, therefore it must be moderate to achieve maximum performance.

If forced-delay is used in case of end-to-end aggregation, then delay is introduced by the sender node only once. Packets are forwarded by the intermediate nodes until they reach their destination. But in hop-by-hop aggregation, forced-delay is added by every node, since packets gets de-aggregated and then aggregated at every hop.

2.3.3 Other Packet Aggregation Techniques

Several packet aggregation techniques for improving throughput on various networks were proposed. Some of them are as follows.

2.3.3.1 Packet Frame Grouping (PFG)

Among one of the first solutions for aggregation was “Packet Frame Grouping” (PFG) [21]. This scheme was devised for improving performance of small packets for MAC protocol in

WLAN network consisting of single hop. Packet Frame Grouping was originally developed for improving multimedia traffic but can be applied to other types of traffic also.

According to this technique, small packets are grouped together having the contention overhead only for the large packet instead for each small packet. This avoids contention overhead between small packets inside the frame, resulting in reduced overhead and increased throughput. This technique is transparent for the receiving node since the structure of packets remains the same. Frame size is limited to certain number of bytes instead to number of packets. Packet Frame Grouping can be seen in the following figure.

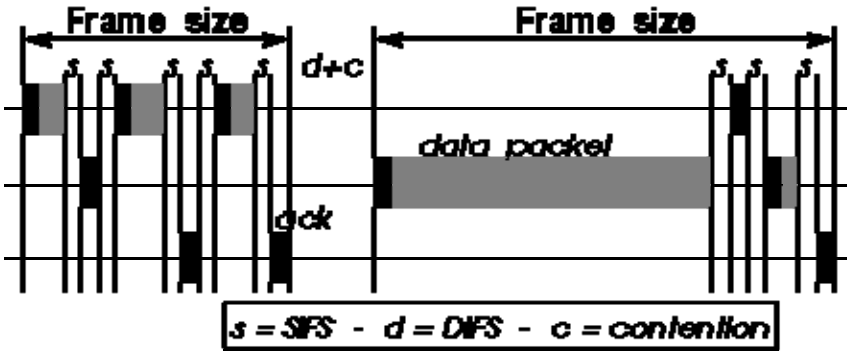


Figure 4: Packet Frame Grouping [21]

Packet Frame Grouping in 802.11 system sends a frame of small packets same as fragmentation [22] sends fragments of a large packet. Packet Frame Grouping works in such a way that MAC includes a counter of number of bytes in current frame. After a successful transmission, size of next packet is added in the transmission queue to current frame size by MAC which is then compared with the maximum frame size. If it is less, this packet is sent after a SIFS (Short Inter-frame Space), otherwise it goes into contention after resetting the counter.

Key benefits of this technique were that packet delivery latency is not increased, no data is copied and also it is not limited to send packets for any particular node.

2.3.3.2 Packet Concatenation at IP-Layer (PAC-IP)

PAC-IP [23] addresses the packet concatenation at IP-level for a wireless LAN having the same next hop. According to this technique, IP packets are concatenated into a single object known as concatenated collection which is then forwarded to link layer from where it is transmitted. This concatenated collection as a whole contains only one header of link layer and physical layer instead of having separate headers for every IP packet.

IP packet size is stored in the IP header and collection size is stored in the MAC header. Both of these values are then used by the receiver node to separate original IP packets from these concatenated packets. Headers remain unchanged both at IP-level and link layer level. PAC-IP concatenation is depicted in the following figure.

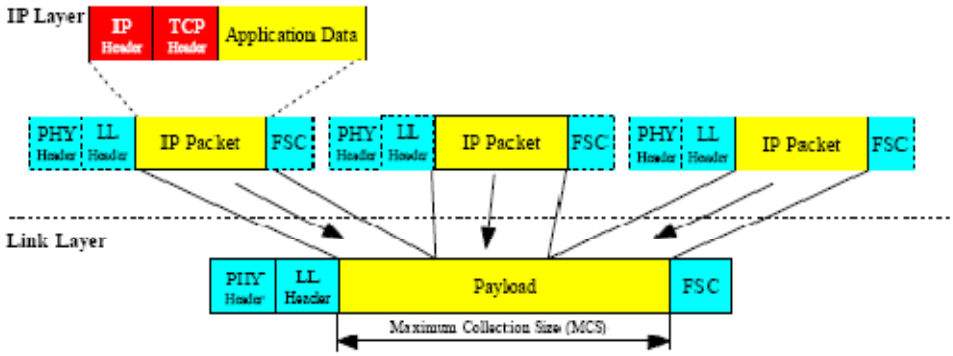


Figure 5: PAC-IP concatenation [23]

It is implemented by inserting a software module below IP layer and above link layer, which concatenates several incoming IP packets into single outgoing collection. Concatenation in this case employs copying data. Data from the next packet is copied to the first arrived packet whose data buffer is extended to maximum size of the collection. Maximum collection size MCS of the packet is adjusted to its optimal value dynamically.

Each collection has a Maximum concatenation time MCT. If a timeout has reached and MCS is still not reached, the collected packet is forwarded immediately for transmission.

Like other approaches, this approach also has some drawbacks like increase in delay which is added during the concatenation phase. In case of MCT expiration, the first packet has delay increased with MCT value.

2.3.3.3 Packet Concatenation (PAC)

Packet Concatenation (PAC) [24] is used at link layer for rate adaptive mobile ad-hoc networks. Main idea of this approach is to concatenate frames at MAC layer to create a super frame which is then transmitted. Packets for concatenation are selected having same next hop address i.e. same for all frames in the collection. Numbers of frames to be concatenated are calculated dynamically by PAC i.e. ratio of current data rate to lowest supported data rate. Up to 9 MAC data frames can be concatenated into a single super-frame. If no more data frames are for same next hop, the current packet is transmitted, if the medium is free to transmit. PAC concatenated super-frame can be seen in the following figure.

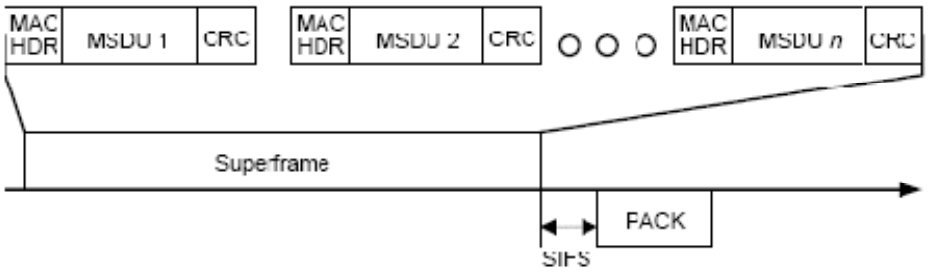


Figure 6: PAC concatenated super-frame [24]

As compared to PFG, PAC showed better overhead reduction without affecting the latency. Concatenated frames are easily de-concatenated by receiver since FSC and MAC header are unchanged.

This approach has also some disadvantages like for frame concatenation, additional data is copied, it is limited to packets destined to particular host i.e. next hop and for PAC, MAC implementation requires to be modified. Also it requires a new packet type for acknowledgement.

2.3.3.4 Adaptive Packet Concatenation (APC)

Adaptive Packet Concatenation (APC) [25] is also a MAC layer packet concatenation scheme for multi-hop ad-hoc networks. Packets are concatenated adaptively into a super packet, having the same next hop, in APC by using the current transmission rate. Payload size is adapted according to the link condition. This super packet is then transmitted. Structure of this super packet is shown in the following figure.

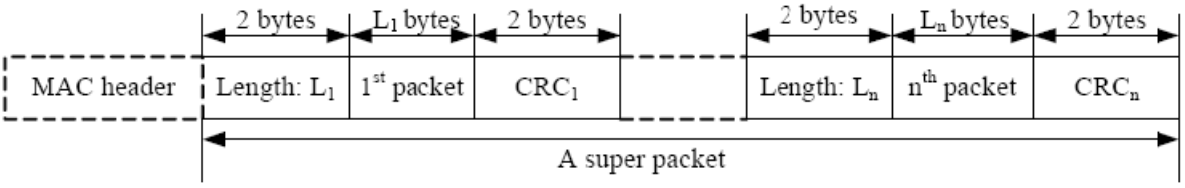


Figure 7: APC super packet structure [25]

A super packet may contain one or more packets. Each data packet in the super packet has three parts i.e. its length, packet itself and an optional CRC field. This length is used by the receiver for splitting super packet into respective small packets. CRC is used to check

packet's integrity against the channel bit errors. Since each packet has its own CRC, therefore individual packets after de-concatenation are checked and in case of any corrupted packet, selective packets need retransmission. In this case, the super packet is reconstructed for the respective corrupted packets. Order of packets in the super packet depends upon their appearance in the queue.

The concatenation threshold is calculated adaptively using the current transmission rate of MAC layer. In general, there are two methods for determining the transmission rate:

1. Transmission rate is determined by the received power of last ACK from next hop in case of a successful last transmission. If this is not the case then the lowest available rate is used.
2. Transmission rate is determined by the received power of CTS frame from next hop.

First method can result in wrong transmission rate value since it depends on the last transmission that could be a poor channel quality or transmission failure. Whereas the second method uses RTS/CTS frames for determining channel quality, which results in calculating more reliable value of transmission rate. Hence, APC uses second method for determining transmission rate.

Link layer concatenation schemes have the ability to adapt their transmission according to the link quality but on the other hand latency is added at each hop because of queuing and data copying. Also the intermediate nodes have to do extra processing resulting in making them slow.

2.3.3.5 IP Based Adaptive Packet Concatenation (IPAC)

IPAC [12] is also an adaptive concatenation protocol that works on the IP layer. This is an end-to-end packet aggregation scheme hence packets are concatenated at the source and de-concatenated at the destination node. Advantage of this scheme is that additional delay added by hop-by-hop packet concatenation is eliminated. Since this approach concatenates packets at source only, therefore, packet size is also calculated only once based on the route quality.

Packets having common destination are concatenated first and then passed to the link layer. Link layer adds its header to this concatenated packet and then it is transmitted. Destination node after receiving this packet de-concatenates it into smaller packets. IPAC packet concatenation can be seen in the next figure.

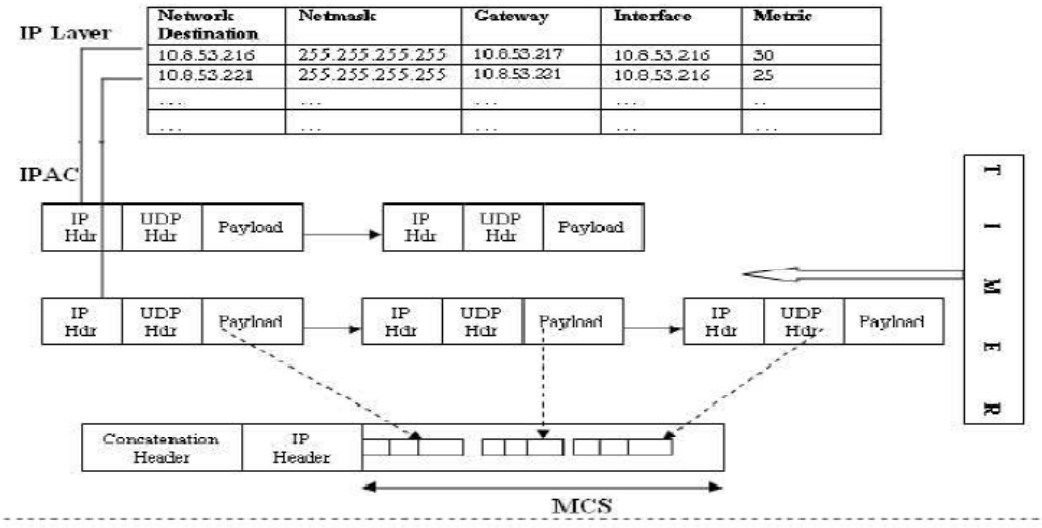


Figure 8: IPAC packet concatenation [12]

Maximum Concatenation Size (MCS) is the size of the concatenated packet which is determined by the route quality to its destination. Maximum Concatenation Interval (MCI) is the time interval for a packet to remain in a queue before concatenation at the sender node. Packets at the source node are delayed by the MCI value where the size of aggregated packet cannot be more than MCS value.

One queue for each destination is maintained by the sender. If size of queue exceeds MCS or MCI expires, packets are de-queued at the sender before delivery. These packets are then added into a single super-packet and a four bytes header is added containing total number of aggregated packets and size of each packet. If a packet size is larger than MCS, then after queue flush, it is transmitted without any further processing.

After reaching the link layer, super-packet is treated as a single IP packet. MAC layer does its processing and then it is transmitted.

The concatenation header is used by the destination node for de-concatenating the respective packet. If the incoming packet does not have any concatenation header, it is passed to the transport layer otherwise it is de-concatenated into small packets which are then delivered to the transport layer individually.

IPAC uses the Weighted Cumulative Expected Transmission Time (WCETT) [26] routing metric for calculating MCS and MCI.

IPAC showed increase in throughput of end-to-end application layer even in high load where it is improved more than twice [12]. In case of low traffic, throughput is improved

slightly since fewer packets are being sent. In case of low traffic, end-to-end delay is also high. IPAC's performance indicates that concatenation is fruitful when the traffic is high.

2.3.4 Packet Aggregation Approach Adopted

Packet aggregation scheme used in this thesis is explained well in [3] [27]. Basically this scheme is designed to improve VOIP quality for wireless meshed networks. This approach can be applied to any routing protocol. Reason behind choosing this approach is that it is recently implemented at my university and every kind of help was available round the clock since there were different technical issues when implemented with MANET game scenario. Also it showed excellent results for VOIP traffic that also consists of small packets [3].

This approach is applied at the network layer by the help of aggregation tunnels built in between the aggregator and de-aggregator nodes. According to this approach, any node can become aggregator de-aggregator.

2.3.4.1 Basic Idea about Aggregation Tunnels and Their Working

In my thesis, I have used aggregation tunnels from the sender node in wireless network to the gateway node. Packets are aggregated at the sender node or gateway only once and then forwarded by the intermediate nodes. Packets that are going out of the wireless network have to pass the gateway, hence they are aggregated at the sender node and transmitted, intermediate nodes on receiving them just forward to next node until they reach the gateway. At gateway, they are de-aggregated into small packets and then transmitted out of the mobile network. Similarly packets that are coming into mobile network first reach the gateway. At gateway, these packets are aggregated according to their destination and then transmitted. After reaching their destination they are de-aggregated by the target node.

2.3.4.2 Aggregation Rules

At the aggregator node, packets having common destination are put into same queues and are marked with a timestamp. As explained in [27], decision about aggregation or to wait depends upon several factors:

- If the queue contains enough packets that can be combined together to make an aggregated packet whose size is larger than the threshold size (600 bytes) and less than Maximum Transmission Unit (MTU) size (1400 bytes), these packets are aggregated and sent without any further delay.

- If forced delay timestamp (10 ms) has expired, these packets are also aggregated and sent immediately even though their size is less than the aggregation size threshold.
- If the parameter ***followup*** is set to ***true***, aggregation algorithm only looks at the age of first packet in the queue and aggregates other packets without considering their age and send them.

In any case other than above, packets will not be aggregated whereas an additional forced delay will be added which can be configured accordingly.

2.3.4.3 Aggregation

First packets are collected in the outbound queue at the MAC layer. Here the packet contains the IP address and the MAC address. These packets are aggregated according to their common destination. After aggregation, a new packet called "IP_META" is created. The aggregated packet consists of small packets having their data and IP-Header. An additional IP-Header is added by the aggregation algorithm to the aggregated packet consisting of 20 bytes identifying it as an aggregated packet. The *protocol* field of IP-Header is set to IP_META. For hop-by-hop aggregation, the aggregation target is next hop whereas in end-to-end aggregation, aggregation target is the end of aggregation tunnel.

2.3.4.4 De-aggregation

A de-aggregator node identifies an aggregated packet by looking at the *protocol* field. If it is IP_META, then it is an aggregated packet. *Length* field of every packet's header contains its packet size which is then used by the de-aggregator for extracting small packets. After extraction, these packets are then passed to the upper layers.

2.3.4.5 Packet Size

Different packet sizes result in varying packet loss ratios. Packet loss can result from several reasons like collisions, route errors, bit errors and queue overflows. Therefore optimum packet size must be chosen to attain less packet loss ratio. Determining suitable packet size that results in less packet loss ratio is rather difficult. Different packet sizes result in different packet loss ratios. Hence an optimum packet size must be adopted to achieve better performance.

To achieve this, two protocols are being used. One is for calculating maximum packet size and the other is for calculating optimum packet size.

Simple Packet Size Selection Protocol

According to this protocol, packet size is determined by sender with the help of information, like channel quality and contention, gathered at the receiver. This protocol works in a way that the receiving node measures SNR constantly of incoming packets, based on this SNR value maximum tolerable packet size is calculated which is then transmitted to the sender node. This maximum packet size is then stored in the routing table of sender node.

Enhanced Packet Size Selection Protocol

SNR helps to calculate the maximum tolerable packet size whereas optimum packet size is also important to calculate i.e. determining level of contention. According to this protocol, SNR is monitored and delivery rate of periodic probe packets is measured, current network load and the link quality is determined at the receiver, pre-calculated table is used for looking optimum packet size for current network state and then pass this optimum packet size value to the sender node by the help of routing messages which is then stored in the routing table.

2.4 Game Traffic Model

Before explaining anything about games, I want to explain a little bit about game server architecture. Commercially games are designed based on any of two architectures i.e. Client Server architecture or Peer to Peer architecture. Peer to Peer architectures have low latencies with the elimination of bottlenecks at server whereas Client Server architectures are rather simple by the implementation point of view and also helpful in retaining game state since the game is controlled centrally.

In Client Server architecture, the game server computes all the game states and then distributes these updated game states to its clients. Clients on receiving these updates, process them and show their effect on the user screen. Clients pass several commands to server like movement and button press. Server after receiving these commands, updates the game state and transmits these updates to the connected clients. The server keeps tracks of all the entities like ammunition, enemies, avatars, map information etc. In case of some explosion, server calculates all effects, physics model calculation etc. and sends these results to the clients. Game server is also responsible for keeping track of game sessions i.e. start of game, end of game, accepts connections from different clients etc. Game client after receiving update packets from server, renders 3D representation locally and displays on the screen. Game server just does the calculation while their 3D representation is done at the client machine.

Basically online games have three main categories i.e. Role Playing Games (RPG), Real Time Strategy (RTS) and First Person Shooter (FPS).

RPG games like “Everquest” [34] have 1000s of players interacting simultaneously. Players can leave or join the game anytime. These games are quite slow in action like player movements and game session updates.

In RTS games like “Age of Empires” [35], players must join the game before it starts and game lasts till someone becomes winner. Unlike RPG games, RTS games have player limit. A particular game session can have at most 30 players. This is because all players must start the game at same time and it will finish when someone wins. Entire game state resides at the game client while game server just sends game updates messages to them.

FPS games like “Quake, Counter Strike” are arcade like games that are full of action. Because of quick action, game states updates are very much important for the players. Game states updates of an FPS game are quite frequent. Game player can join or leave a particular game session. Since the game server overhead for these types of games is quite high, therefore numbers of game players are limited i.e. about 30.

In this thesis, I have studied Quake3, a popular interactive online FPS game. Like other most of the FPS games, Quake3 also has central server client architecture. Game server maintains the game states i.e. current map info, game players, kills, statistics of players etc. Game clients connect with the server. Server transmits periodically current states of game to the clients.

In this section, first of all I would like to explain a more general traffic model of an FPS game [29]. After that, I will explain the traffic model of Quake3 as presented earlier in [4].

2.4.1 General Traffic Model of First Person Shooter Game

Traffic model of any game explains its traffic behavior i.e. effect of user interactions, effect on game traffic by the increase in players, delay, jitter, packet loss and link quality for better performance, packet length, inter-arrival time etc. Good knowledge about game’s traffic model helps greatly to improve performance and achieve better quality of service. There are some main points that need to be considered while defining a general traffic model for a first person shooter game as explained in [29], such as packet rates, packet sizes etc in both directions i.e. from game server to game clients and game clients to game server.

2.4.1.1 Protocol Design

In FPS games, because of their fast action and continuous fighting nature, a game player is in desperate need to know about certain up-to-date information about other players like player locations, scores, movements, actions etc. This information needs to be updated and distributed at a high rate periodically. To achieve this, a communication protocol needs to be built, responsible for exchanging this information continuously among the game players.

Most of the FPS games are designed with client server architecture where the game server runs on a machine attached to the internet and different game clients communicate with this server. Individual player's actions from every client are transmitted to the server. Then server process these actions received from all players and transmit the current game state to every client. This game state is crucial for every player.

Designing such protocol requires some key points to be considered:

- To support maximum number of players, traffic must be minimized i.e. minimizing the bandwidth required for communication between clients and server.
- Accurate and timely copy of game state must be delivered by server, at the same time, to each client. Similarly, each client must send its current state frequently to the server so that it can be processed and transmitted to all other players.

2.4.1.2 Game Model

As far as the game model is concerned, after designing such a protocol, game clients will deliver just small actions to servers, instead of their complex graphic details. In response server will also transmit these little actions along with the consequences, which are also in a simple form, to other players related to same field-of-view. Game clients generate graphics accordingly to the received code.

Packet length from client to server is small since only actions are being delivered. Numbers of actions are limited in a particular game like according to [32] there are less than 20 actions in Quake3. Typical actions are fall down, dive, jump, pick up weapon, turn left, turn right etc. Messages from server to client are also not very large since they only contain the updates which are processed accordingly on the client machine and their 3D rendering is displayed.

Similarly, inter-arrival time of packets, containing game state, from server to clients need to be fixed and at some minimum rate. Inter-arrival time of packets from clients to server also need to be fixed and at minimum rate [29].

Upon engaging a combat between players, game server needs to process and transmit extra information as compared to the normal state.

Message length from server to client is directly proportional to the number of players playing at that time. Since server processes information gathered from all clients and then transmits to every client, therefore in case of more players, more information is gathered, processed and transmitted resulting in increased packet size.

2.4.2 Synthetic Traffic Model of Quake3

A synthetic traffic model for Quake3 is well explained in [4]. For this thesis study, I am following the same traffic model as described in [4] therefore in this section I am going to explain from [4].

Since Quake3's game traffic is a real-time traffic and special measures regarding quality of service have to be carried out for delivering real-time traffic.

Because of fast action and limited decision time in Quake3, the underlying network has a great impact on game players. In [4], the authors have carried out several experiments over LAN environment for Quake3 for about 2 months having 2 to 8 players to observe game traffic and devised game traffic model.

Observations and the devised traffic model for Quake3 from [4] are as follows.

2.4.2.1 Server to Clients Packet Length

According to [4], packets length from server to game clients greatly depends on number of players of game at that time and minor on the map played. Increase in number of players results in increase in packet size.

2.4.2.2 Client to Server Packet Length

Packet length from client to server is independent of number of players or map. Packet size does not vary so much. Sizes of these packets were in the range of 50 bytes to 70 bytes.

2.4.2.3 Server to Client Packet Inter-Arrival Time

Server sends game state packets to every client regularly. This time is independent of number of players or map of game. Server sends an update packet almost after every 50 ms approximately. If suppose, 20 game clients are connected to the server, a burst of 20 packets will be sent after every 50 ms from server.

2.4.2.4 Client to Server Packet Inter-Arrival Time

Game traffic from client to server depends upon some factors like the game map being played and the client's graphics card. If client's graphics card is more powerful, i.e. has bigger RAM and fast graphics processor, it will send more packets as compared to the clients having small graphics card since their processing speed is higher. In [4], different experiments were performed on three types of systems having different specification. For my simulations, I have chosen "Graphics Card 1" having 32 MB RAM. Specification of this computer as described in [4] is DELL GX260 (2 GHz processor, 256 MB RAM, 32 MB Intel 845G graphics card). A packet is transmitted to server after every 10.75 ms approximately.

The second graphics card, named "Graphics Card 2", is also a 32 MB graphics card but with less processor. Specification of this system as described in [4] is Compaq EV500 (1.6 GHz processor, 256 MB RAM and 32 MB Nvidia GeForce2 MX/MX 400 graphics card). Output of this graphics card shows two peaks. One significant peak at 10.75 ms and the second one at 11.75 ms with packets span till 25 ms apart.

2.4.2.5 Server to Client Packets per Second (PPS) and Data Rates

Since, as stated earlier that server sends an update packet to every client after approximately 50 ms, therefore packet per second rate from server to every client is around 19 to 20 packets per second. Also, packet size from server to clients depends on number of clients, therefore data rate also changes, as it is also dependent on the size of packet and to a lesser extent on the map played.

2.4.2.6 Client to Server Packets per Second (PPS) and Data Rates

As the graphics card and played map affects the packet inter-arrival time, so as the case with packets per second rate from client to server. Higher graphics cards, because of their increased RAM and high processing, send more packets in a second resulting in high packets per second rate. As far as the data rate is concerned of clients, it corresponds to be same as PPS rate of client i.e. fluctuates according to the graphics card and game map. Data rate for Graphics Card 1 is between 40 and 45 kbps and for Graphics Card 2 is between 35 and 40 kbps.

2.5 Packet Aggregation and Quake3 Game Model

Packets generated at the game client are aggregated into bigger IP_META packet according to their common destination. An aggregation tunnel is created between game client and the

gateway node. Destination of IP_META packet is the end of aggregation tunnel i.e. wireless gateway node instead of the game server. After reaching at gateway, IP_META packets are de-aggregated into small game packets (QUAKE3P), which are then forwarded to the fixed network and finally to the game server.

Traffic from game server first reaches at the gateway node in the wireless network. These packets are then aggregated according to their common destination into IP_META packets and their destination is set to the wireless node destined by the game server. After reaching at the mobile node, these packets are de-aggregated into smaller game packets i.e. QUAKE3P.

Aggregation tunnels are being created only in wireless network.

2.6 Network Simulator 2 (NS-2)

In order to carry out various simulations and obtaining results **Network Simulator** [5] [28] is used. The version which I used for simulations is (NS-2.26). In this section I am going to give a brief general introduction to NS.

Basically Ns-2 is an object oriented discrete event Network Simulator, written in C++ having OTCL (object oriented TCL) as a frontend interpreter [5]. Ns provides simulation support for TCP, routing algorithms, multicast protocols over wired and wireless, local and satellite networks [30]. A simulation consists of events and timings of events are maintained by a scheduler. Because of its open source model, it is extensible and heavily popular in carrying out simulations for ad-hoc networks research.

Network topology with simulation setup and configuration is written in TCL by the user which is then simulated by NS with all the specified parameters. OTCL is used because configuration and topology is fast to write and easy to change. Network elements in NS are developed as object-oriented classes. New simulator objects are created by users in OTCL which are then mirrored by corresponding objects in C++. C++ does the per packet processing. It is fast to run, and has complete control over objects and classes.

2.6.1 Basics for Using NS

Following are the basic points to consider in using NS:

- **Create a TCL script** containing the network topology, nodes, traffic sources, sinks, links, output file(s) etc.
- **Specify parameters for objects** like for nodes, link speeds, queue sizes, traffic models, transport protocol, traffic type, simulation time etc.

- Modify NS source code, if necessary.
- Don't forget to save all the simulation **output in trace file(s)** so that post processing can be done on them in order to get meaningful results.
- Run simulation multiple times with different seeds in order to get the confidence interval.

2.6.2 Creating a Basic NS Simulation

Following are the steps involved in creating a basic simulation:

1. Create simulator object and event scheduler.
2. Create corresponding nodes and links.
3. Create connections between nodes.
4. Create traffic between sources and sinks.
5. Enable tracing for storing output.

Now I am going to explain them a little bit with examples [31]. Detailed information can be found at [5].

2.6.2.1 Create Simulator object and Event Scheduler

1. Create a simulator object
    a. set ns [new Simulator]
2. Schedule event(s)
    a. $ns at <time> <event>
        i. <time> : specified in seconds.
        ii. <event> : any TCL event i.e. start, stop etc.
3. Start scheduler
    a. $ns run

2.6.2.2 Create Corresponding Nodes and Links

1. Create nodes
    a. set n0 [$ns node]
    b. set n1 [$ns node]
2. Create links for connecting nodes
    a. $ns duplex-link $n0 $n1 <bandwidth> <delay><queue type>
        i. <bandwidth> : specify bandwidth of link in Mb.
        ii. <delay> : specify propagation delay in Ms.

iii. <queue type> : specify any of the queue type i.e. DropTail, RED, CBQ, FQ, DRR, SFQ.

2.6.2.3 Create Connections

Transport connections protocols are created here like TCP, UDP, multicast etc with are then attached to the nodes.

1. UDP transport channels source and sink are created. Traffic is generated from source and terminated at sink.
    a. set udp_src [new Agent/UDP]
    b. set udp_dst [new Agent/NULL]
2. These are then attached to nodes that will generate traffic and then connected with each-other.
    a. $ns attach-agent $n0 $udp_src
    b. $ns attach-agent $n1 $udp_dst
    c. $ns connect $udp_src $udp_dst

2.6.2.4 Create Traffic Between Sources and Sinks

Application objects are attached to transport protocol objects and then traffic is generated. CBR (Constant Bit Rate) is used for UDP traffic and then traffic parameters are set.

1. set cbr [new Application/Traffic/CBR]
2. $cbr set packetSize_ 1500
3. $cbr set interval_ 0.05
4. $cbr attach-agent $udp_src
5. $ns at <time> “$cbr start”

In line-1 of the above code, “cbr” traffic source is created. In line-2, the packet size is set to 1500 bytes. In line-3, time interval is mentioned in seconds after which next packet is generated. In line-4, the traffic source is attached to UDP agent that will generate the traffic. And in the last line, time is mentioned in seconds at which the CBR traffic will start generating.

2.6.2.5 Enable Tracing for Storing Output

It is important to save the simulation output in a file. Post scripting in Perl/awk is applied to these files that enables to extract meaningful information. Syntax for specifying a tracefile is as follows.

1. set tracefile [open <tracefile name> w]

a. <tracefile name> : specify any name for trace file.

2. $ns trace_all $tracefile

2.6.3 Trace File Format

Trace files contain all the simulation output generated. The output of a trace file is somewhat similar to the one explained below in the example. Each row has several columns and every column has a meaning. First column specifies the event, next field specifies the time at which that event has occurred, next two columns contains the “from node” and “to node” in between that event has occurred, next column tells the packet type, after is the size of the packet, “-------” are known as packet flags, after them is the column of flowid. Since given simulation can contain several flows, therefore every flow has its own id. Proceeding two columns specify the source and destination nodes addresses. The second last field is of sequence number of packet and the last field is of packet’s ID.

Trace file format for a basic wired network simulation is as follows.

Event	Time	From	To	Pkt_type	Size	---	Flow_Id	Src	Dst	Seq_No	Pkt_Id
+	1	0	2	udp	1500	---	1	0.0	2.0	10	5
-	1	0	2	udp	1500	---	1	0.0	2.0	10	5
r	1.007	0	2	udp	1500	---	1	0.0	2.0	10	5

Table 2.1: Wired Trace Format

Where events are of four types:

- + enque.
- - deque.
- r receive.
- d drop.

Wireless trace file format and wired-cum-wireless trace file formats have several additional columns as compared to the above example. They are well explained in [5].

Wired-cum-wireless simulations contain wired and wireless network scenario. Traffic is generated between nodes of both sides. When a packet is sent or received in wired network, trace file records entries something like explained earlier in table 2.1. Trace format for wireless scenario is explained below.

```
s 17.631283910 _117_ AGT  --- 31740 QUAKE3P 65 [0 0 0 0] ------- [0.0.74:0
1.0.1:1 32 0]
```

Table 2.2: Wired-cum-wireless trace format.

Above table shows the trace format of a wired-cum-wireless simulation for a single packet. Following table explains the individual fields in detail.

Field	Explanation
`s`	Event. There are 4 events. s: sent, r: received, f: forwarded, d: drop.
`17.631283910`	Time at which this event is occurred.
`_117_`	Node ID.
`AGT`	Trace name. AGT: agent trace, RTR: router trace.
`---`	Reason, usually when a packet is dropped.
`31740`	Packet ID.
`QUAKE3P`	Packet type.
`65`	Packet size.
`[0 0 0 0]`	Hexadecimal values, [time, destination MAC, source MAC, packet type code identifier]
`-------`	Flags.
`0.0.74:0`	Source IP address with port number.
`1.0.1:1`	Destination IP address with port number.
`32`	Time to live value.
`0`	Number of times packet is forwarded.

Table 2.3: Wired-cum-wireless trace format explanation.

Further explanation of these fields can be found in [5].

3 Design and Implementation

In this chapter I am going to explain the setup design and afterwards implementation in NS for analyzing the online game traffic in hybrid mobile ad-hoc wireless networks with and without packet aggregation.

3.1 Simulations Goal

The basic goal of my simulations is to analyze the behavior of online game traffic of Quake3 played in internet connected mobile ad-hoc wireless network with and without packet aggregation and calculate delay, jitter and packet loss.

There were a series of simulations carried out falling in two main categories, one is with packet aggregation and the other without packet aggregation. Simulations results are obtained by increasing/decreasing game clients with varying hops in between game-clients (wireless network) and their internet connected gateway.

SIP traffic consisting of four traffic flows is constantly being generated in between several nodes from [33]. Three flows are between wireless nodes and one is in between wired and wireless nodes. Exponential traffic flows are also being generated, in order to create network load. They are just for creating background traffic. Therefore the network consists of game traffic, SIP traffic and exponential traffic.

3.2 Step-wise Distribution

Following steps are carried out to get the final results.

3.2.1 TCL scripting

In order to carry out different simulations in NS, TCL scripting is done. A TCL file contains scripts for node creation, node configuration, simulation scenario, routing protocol, links between nodes, nodes communication, traffic flows, output trace files, MAC layer settings, simulation run time etc.

Each simulation has its own separate TCL file consisting of particular scenario and settings.

3.2.2 C-Language Coding

For the game traffic, a new packet type is generated. These game packets are named as "QUAKE3P" packets.

Packet aggregation/de-aggregation code from [3] is adopted which is changed to operate according to the online game scenario with the new packet type. Also, this aggregation is neither adaptive nor hop-by-hop. The game packets are aggregated at sender node and de-aggregated at the gateway node from where they are forwarded towards the game server. Same case is for the incoming packets i.e. incoming packets from game server first arrive at the gateway where they are aggregated and then de-aggregated after reaching at the destination node.

3.2.3 Post Scripting

After running simulation, a trace file is generated by the TCL script. This trace file records all the events occurred during the simulation run-time, bundled with all the necessary details. In order to extract the meaningful information from these trace files and calculate different parameters, post scripting is done.

For post scripting, mostly Perl or AWK is used. For my simulations, I have used Perl for post-processing trace files.

3.2.4 Graphs

After obtaining meaningful information by running Perl scripting on trace file, graphical representation of this information is done by drawing graphs in "Microsoft Excel". Comparison among simulations becomes a lot easier by these graphs.

3.3 Implementation of Quake3 Traffic in Hybrid MANET with/without Packet Aggregation in NS(2.26)

In this part of the chapter, I will explain stepwise the implementation and design of a simulation in NS.

Implementation of Quake3 game traffic with and without packet aggregation in a particular scenario for wired and wireless nodes is as follows. Each step is later explained in respective subsections.

1. New packet type.
2. Aggregation queue.
3. De-aggregator agent.

4. Quake3 game model code.
5. Simulation parameters.
6. Network topology.
7. Nodes creation.
8. Connections between nodes.
9. Setting game server.
10. Setting game client.
11. Scheduling events.
12. Starting simulation.

Explanation of these points with the sample code from simulations Tcl script is explained as follows.

3.3.1 New Packet Type

There are different types of packets flowing through the network during simulation runtime like **exp, udp, encapsulated** etc but among them, two packet types are of more importance. One packet type is from [3] which is the type of aggregated packets named as “IP_META”. The other packets are of game traffic that is assigned a new packet type known as “QUAKE3P”. Code snippet of creating a new packet type in NS is as follows.

Header of this new packet is defined as follows. Name of the file from which this code has taken is “quake3pkt.h”.

```
01:   struct hdr_quake3p {
02:        static int offset_;
03:        inline static int& offset() { return offset_; }

04:        inline static hdr_quake3p* access(const Packet* p) {
05:                  return (hdr_quake3p*) p->access(offset_);
06:   }

07:   int size() { return size_; }
```

Code 3.1: Header of Packet QUAKE3P

Following is the code for new packet type QUAKE3P taken from file “quake3pkt.cc”. This new packet type will represent the QUAKE3 game traffic in both directions i.e. to and from game server and clients in the trace file.

```
01:   int hdr_quake3p::offset_;
02:   static class quake3pHeaderClass : public PacketHeaderClass {
03:   public:
04:        quake3pHeaderClass() : PacketHeaderClass("PacketHeader/Quake3p",
05:                                        sizeof(hdr_quake3p)) {
06:                    bind_offset(&hdr_quake3p::offset_);
07:        }
08:   } class_quake3phdr;
```

Code 3.2: Code for new packet type QUAKE3P

This new packet type is then added in the "packet.h" file. Because of this, all the game traffic generated between nodes will have the packet type "QUAKE3P". When several game packets are aggregated together, a new packet is formed having type "IP_META" that implements the encapsulation technique of AODV-UU.

3.3.2 Aggregation Queue

Packet Aggregation is implemented at Aggregation Queue [3]. It consists of two methods i.e. enque and dequeue. Enque puts the incoming packet in queue and deque implements aggregation algorithm to queued packets.

3.3.2.1 Enque

Enque method (see Code3.3) takes the incoming packet, sets the timestamp of incoming packet to current time (line 05) and then stores it in the queue (line 07). If length of queue exceeds queue limit, next incoming packets are dropped (lines 08-11).

Following code is en-queuing EXP packets (VOIP traffic) and QUAKE3P packets (game traffic) that can be seen on line 04.

```
01:   void Aggregator::enque(Packet* p)
02:   {
03:     struct hdr_cmn *ch = HDR_CMN(p);
04:     if (ch->ptype() == PT_EXP || ch->ptype() == PT_ENCAPSULATED || ch-
>ptype() == PT_QUAKE3P) {
05:      ch->ts_ = Scheduler::instance().clock();
06:     }
07:   q_->enque(p);
08:     if (q_->length() > qlim_) {
09:      q_->remove(p);
10:      drop(p);
11:     }
12:   }
```

Code 3.3: Enque function

3.3.2.2 Deque

Here in this function, actual aggregation algorithm is implemented (see Code 4). This function takes packets from queue. Each queue contains packets having same destination. Hence packets are aggregated with respect to the first packet in aggregation queue. According to the following code, top most packet is taken from the queue (line 03) and timestamp is set to the current time (line 04). Actual queuing algorithm begins at line 05. Packets of type EXP (VOIP) and QUAKE3P (quake3 game) will be aggregated.

Size of combined packets in IP_META packet must be less than $SIZE_{min}$ threshold, otherwise a new IP_META will be formed. Size of IP_META packet will never exceed $SIZE_{max}$ threshold and MTU (maximum transmission unit) which can be configured in the Tcl file. In my simulations, MTU is set to 1400 bytes.

```
01:   Packet* Aggregator::deque()
02:   {
      ......
03:     p_top =  q_->lookup(0);
04:     currenttime = Scheduler::instance().clock();
      ......
05:     if ((hdr_cmn_packet->ptype() == PT_EXP || hdr_cmn_packet->ptype()
== PT_ENCAPSULATED || hdr_cmn_packet->ptype() == PT_QUAKE3P))
06:     {
      ......
07:     switch (numpacks) {
08:       case 0: {
09:           if (debug_output_) printf("Aggregator @%d:: No packets to
aggregate\n", currentaddr);
      ......
10:       }
11:       case 1: {
12:           if (debug_output_) printf("Aggregator @%d:: Only one packet,
do not aggregate\n", currentaddr);
      ......
13:       }
14:       default: {
15:           if (debug_output_) printf("Aggregator @%d:: Aggregated %d
packets\n", currentaddr, numpacks);
      ......
16:       }
17:     }
18:    }
19:   }
```

Code 3.4: Dequeue function

3.3.2.3 Aggregation Algorithm Explanation

Aggregation algorithm is well illustrated in the flow-chart diagram as shown below. According to this algorithm, first size of packets to be aggregated is determined having same destination as first packet P_0. If the packet size is equal to or greater than $SIZE_{min}$ then all packets will be aggregated in to IPMETA packet having same next hop as first packet up to limit of $SIZE_{max}$ and it will be sent. If the size of packets to be aggregated is less than $SIZE_{min}$ then packets older than T_{delay} will be aggregated into an IP_META packet whose size must not exceed $SIZE_{max}$.

An IP_META packet is formed when there are two or more than two packets older than T_{delay} (see Code 4, lines 14-15). In case of a single packet that is older than T_{delay}, then it will be sent as it is without aggregation (see Code 4, lines 11-12). If no packet is older than T_{delay}, then nothing will be sent and delay timer will set (see Code 4, lines 08-09). If the **followup** parameter is set “true”, then packets in the queue will be aggregated according to the age and destination of first packet P_0.

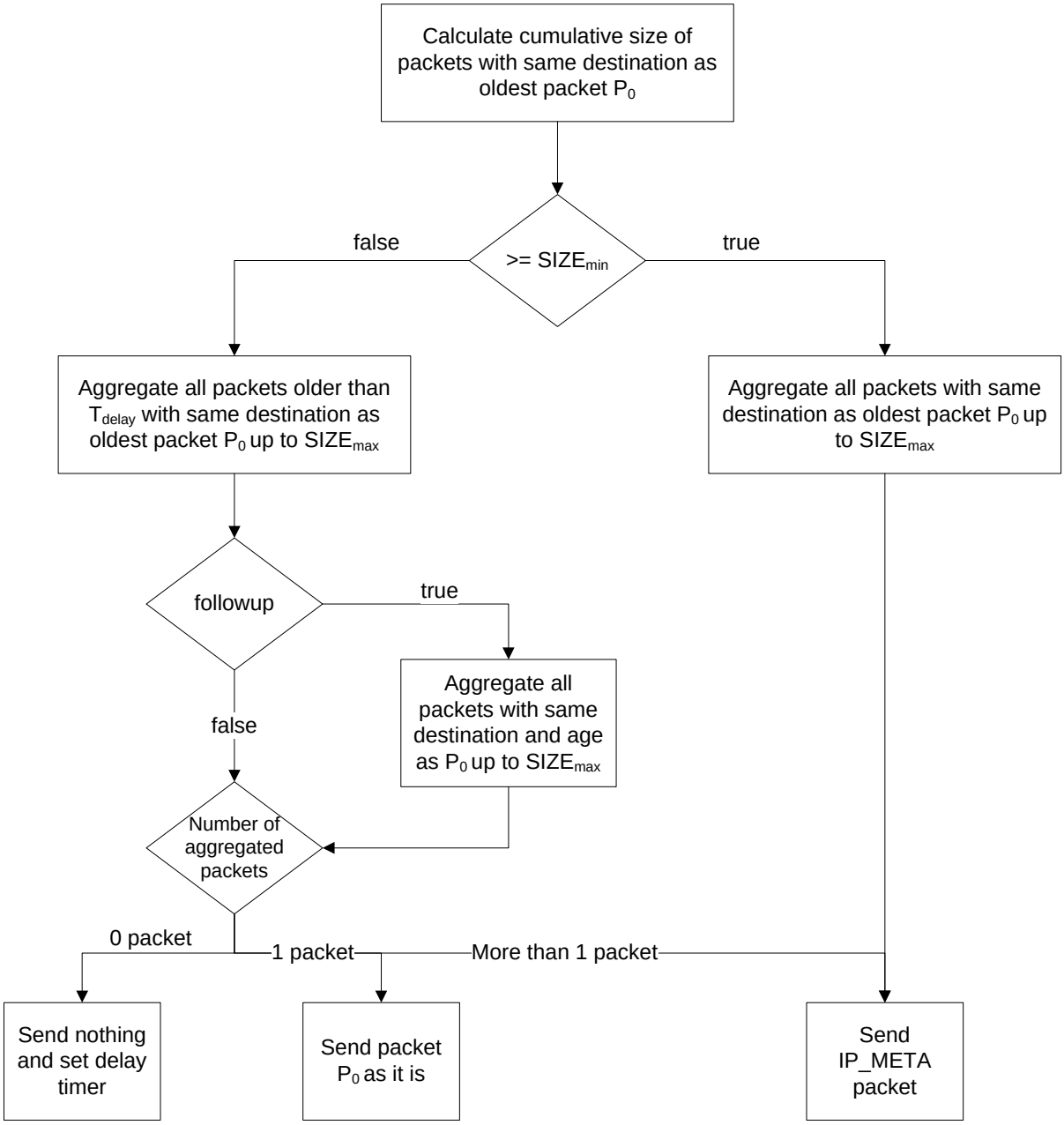


Figure 9: *Aggregation Algorithm*

3.3.2.4 Simulation Variables for Aggregation

Aggregation queue is configurable from Tcl script. Values of different parameters can be changed before creating any node. Required syntax is shown in the following table. It consists of three parts. First part is written before configuring any of the variable, second part consists of the variable, whereas the last part contains the desired configuration value.

Required Syntax	Variable	Value
`Queue/Aggregator set`	`debug_output_`	`false`

Table 3.1: Queue/Aggregator Configuration Syntax

Five types of variables can be configured through Tcl script for Aggregation/Queue. List of these variables with little description is mentioned in the following table.

Variable	Description
`debug_output_`	• "true", displays debug output of Aggregation on the console. • "false", no debug output is displayed.
`agg_followup_`	If "true", aggregates all packets having same age and destination w.r.t. first packet up till $SIZE_{max}$.
`aggthreshold_`	Configuration for $SIZE_{min}$, mentioned in bytes.
`mtusize_`	Maximum transmission unit size for MAC protocol, also mentioned in "bytes".
`max_delay_`	Configuration for T_{delay}, mentioned in "ms".

Table 3.2: Queue/Aggregator Configuration Variables

Respective code in the Tcl simulation script is as follows.

```
01:   Queue/Aggregator set debug_output_ false
02:   Queue/Aggregator set agg_followup_ true
03:   Queue/Aggregator set aggthreshold_ 600
04:   Queue/Aggregator set mtusize_ 1400
05:   Queue/Aggregator set max_delay_ 10
```

Code 3.5: Queue/Aggregator simulation code

3.3.3 Agent/Deaggregator

In order to de-aggregate packets, Agent/Deaggregator is used. The aggregated IP_META packets are received by Agent/Deaggregator which are forwarded to node after de-aggregation.

C-language code files [3] for Agent/Deaggregator consists of two files namely "agent-deaggregator.h" which is the header file and "agent-deaggregator.cc" containing the required code for de-aggregation.

In order to attach Agent/Deaggregator to a node, following syntax is used in Tcl simulation file.

```
$node attach [new Agent/Deaggregator] 58
```

Code 3.6: Code for attaching deaggregator agent to a node

According to the above mentioned code, "`$node`" is the name of node to which Agent/Deaggregator is to be attached and "58" is the numerical representation of IP_META packet.

3.3.4 Quake3 Game Model Code

Code for Quake3 game model [4] consists of two files. One of game server named as "Quake3Server.cc" and second of game client known as "Quake3Client.cc".

3.3.4.1 Game Server

Packet length from game server to its clients depends upon two factors i.e. number of players and map being played. Since the impact of played map is quite less therefore it is not considered in this implementation [4].

Sample code of game server packet length calculation and Inter-arrival time is given below.

```
01:        size_ = int (Random::lognormal(79.340543, 0.24507092));
           /* plus a negative exponential with mean 13 for every additional
player */
02:        for (int i=3; i<=nrOfPlayers; i++)
03:           size_ += int (Random::exponential(13));
           /* send one packet to each player */
04:        for (int i=1; i<=nrOfPlayers; i++)
05:                    send(size_);
           /* figure out when to send the next one */
06:        nextPkttime_ = next_interval(size_);
07:        timer_.resched(interval_);
```

Code 3.7: Server packets length calculation

The packet size from server to clients depends on number of players. It is calculated as the base size distribution for two players plus a negative exponential with mean 13 for every additional player. Base packet size for two players is calculated by taking lognormal with mean 79.340543 and standard deviation 0.24507092 (code 3.7, line-1).

Also the inter-arrival time between simulated packets is 50 ms, which in actual is a gamma distribution having peak at 50 ms.

```
01:   Quake3Server::Quake3Server()
02:   {
03:        //Quake3 server transmits pks every 50 ms
04:        interval_ = 0.05;
05:        bind("nrOfPlayers_", &nrOfPlayers);
06:   }
```

Code 3.8: Server packets Inter-arrival time

3.3.4.2 Game Client

Packet length, in actual, from game client to server is assumed to be too complex for simulation [4], therefore a rough approximation is implemented (see code 3.9 line 01). Around 60% packets are transmitted having interval of 01.75 ms (lines 03, 06), whereas the remaining with an exponential distribution having mean 4.29 (lines 08, 10). Two types of graphic cards were used in simulations i.e. graphic card 1 and 2. For the simulation "gc 1" is used which depicts most modern 32 Mb graphics card.

Code of game client is given below.

```
01:        size_ = int (Random::normal(64.151757 , 3.2035755));
02:        double chooseDist = Random::uniform();
03:        if ((gc == 1) && (chooseDist <= 0.6))
04:             return(0.01075);
05:        else if ((gc == 2) && (chooseDist <= 0.16))
06:             return(0.01075);
07:        else if ((gc == 1) && (chooseDist > 0.6))
08:             return(Random::exponential(4.29)/1000+0.01075);
09:        else if ((gc == 2) && (chooseDist > 0.16))
10:             return(Random::exponential(5.85)/1000+0.01075);
11:        nextPkttime_ = next_interval(size_);
12:        timer_.resched(nextPkttime_);
```

Code 3.9: Game Client packet distribution

3.3.5 Simulation Parameters

Simulation parameters consist of three parts.

- Global variables.
- MAC layer parameters.
- Physical layer parameters.

3.3.5.1 Global Variables

For convenience, certain variables are declared and given values in start of every simulation instead of hard-coding them in the code. By this approach, variables become easily accessible and simulation can be controlled by changing their values. Also, the code becomes easily understandable and self-explanatory.

Following code snippet is taken from one of the Tcl simulation files in order to explain how they are used in the simulations.

Simulation Variables	Values	Description
set val(chan)	Channel/WirelessChannel	Channel type
set val(prop)	Propagation/TwoRayGround	Radio propagation model
set val(netif)	Phy/WirelessPhy	Network interface type
set val(mac)	Mac/802_11	MAC type
set val(ifq)	Queue/DropTail/PriQueue	Interface queue type
set val(ifqa)	Queue/Aggregator	Aggregator nodes queue type
set val(ant)	Antenna/OmniAntenna	Antenna model
set val(ifqlen)	50	Maximum packets in IFQ
set val(nn_wireless)	100	Number of mobile nodes
set val(rp)	AODVUU	Routing protocol
set val(bl)	2200	Border length of topology
set val(runtime)	101	Simulation runtime
set val(nn_wired)	43	Number of wired nodes
set val(nn_bsnodes)	1	Number of base station nodes
set val(tracefile)	"sim_111_2h8c_agg.tr"	Name of trace file
set val(G729Rate)	12.8k	VOIP rate
set val(G729Pktsize)	32	Packet size
set val(flows)	4	Number of call flows
set val(rttrace)	OFF	Route trace is off
set val(agttrace)	ON	Agent trace is on
set val(mactrace)	OFF	MAC trace is off

Table 3.3: Global Variables of simulation

3.3.5.2 MAC Layer Parameters

802.11g layer is used in the simulations. MAC layer is configured for 24-Mbps for the simulations. It consists of different parameters. Syntax for MAC layer configuration is as follows.

Required Syntax	Variable	Value
Mac/802_11 set	basicRate_	24Mb

Table 3.4: MAC layer configuration syntax

From the above table, “required syntax” will be written with every variable for setting value. Other MAC layer variable values with little description are stated in the next table.

MAC layer Variables	Values	Description
basicRate_	24Mb	Bandwidth rate of MAC
dataRate_	24Mb	Data transfer rate for traffic
CWMin_	15	Sets minimum contention window to 15
CWMax_	1023	Sets maximum contention window to 1023
SlotTime_	9us	Sets slot time for back-off window
CCAtime_	3us	Sets CCA time
SIFS_	16us	Sets short inter-frame space
PreambleLength_	96	Sets length of preamble in bits
PLCPHeaderLength_	40	Sets Physical Layer Convergence Protocol header length in bits
PLCPDataRate_	6e6	Sets Physical Layer Convergence Protocol rate to 6×10^6 Mbps
ShortRetryLimit_	7	Sets long retries
LongRetryLimit_	4	Sets short retries

Table 3.5: MAC layer Variables

Sample code as stated in the Tcl simulation file is as follows.

```
01:   Mac/802_11 set basicRate_ 24Mb;
02:   Mac/802_11 set dataRate_ 24Mb;
03:   Mac/802_11 set CWMin_ 15
04:   Mac/802_11 set CWMax_ 1023
05:   Mac/802_11 set SlotTime_ 9us
06:   Mac/802_11 set CCAtime_ 3us
07:   Mac/802_11 set SIFS_ 16us
08:   Mac/802_11 set PreambleLength_ 96
09:   Mac/802_11 set PLCPHeaderLength_ 40
10:   Mac/802_11 set PLCPDataRate_ 6e6
11:   Mac/802_11 set ShortRetryLimit_ 7
12:   Mac/802_11 set LongRetryLimit_ 4
```

Code 3.10: MAC layer configuration code

3.3.5.3 Physical Layer Parameters

Simulations scenario used is well defined in [33]. This scenario is designed with nodes having carrier sense range of 550 meters and transmission range of 250 meters. It is designed with the following most commonly used parameters for 802.11g channel.

Property	Value
Transmission rate	24 Mbps
Node spacing	200 meters
Carrier sense range	550 meters
Transmission range	250 meters
Transmission power	0.1 Watt
Antenna height	1.5 meters
System loss	1
Sensitivity	-85 dbm

Table 3.6: Simulation Scenario Characteristics

Physical layer configuration consists of certain values i.e. “carrier sense threshold” and “receive threshold” that need to be calculated by previously provided carrier sense range and transmission range. In NS, these values are calculated by the help of “threshold” command.

Syntax of such code is given below.

```
01:   ./threshold -m TwoRayGround  -Pt 0.1 -fr 2.4e+9  -L 1 550
02:   ./threshold -m TwoRayGround  -Pt 0.1 -fr 2.4e+9  -L 1 250
```

Code 3.11: Calculation of Physical Layer Parameters

Line 01 in the above code calculates the “carrier sense threshold” known as **CSThresh** whereas line 02 calculates “receive threshold” known as **RXThresh** of physical layer. These values are then used in the simulation for configuring physical layer as shown in the next code sample.

Required Syntax	Variable	Value
`Phy/WirelessPhy set`	`CSThresh_`	`5.53241e-12`

Table 3.7: Physical layer configuration syntax

Other physical layer variables with their configuration and little description are given below.

Physical layer Variables	Values	Description
CSThresh_	5.53241e-12	Carrier sense range for 550 meters calculated by code 3.9 line 01
RXThresh_	1.296e-10	Receive range for 250 meters calculated by code 3.9 line 02
bandwidth_	24Mb	Bandwidth of physical layer link
Pt_	0.1	Node's transmission power, mentioned in Watt
freq_	2.4e+9	Frequency of network, set to 2.4 GHz
L_	1.0	System loss factor
CPThresh_	10.0	Capture threshold

Table 3.8: Physical layer viable with description

Sample code from Tcl simulation file is shown in next code sample.

```
01:   Phy/WirelessPhy set CSThresh_ 5.53241e-12
02:   Phy/WirelessPhy set RXThresh_ 1.296e-10
03:   Phy/WirelessPhy set bandwidth_ 24Mb
04:   Phy/WirelessPhy set Pt_ 0.1
05:   Phy/WirelessPhy set freq_ 2.4e+9
06:   Phy/WirelessPhy set L_ 1.0
07:   Phy/WirelessPhy set CPThresh_ 10.0
```

Code 3.12: Physical layer configuration code

3.3.6 Network Topology

After setting variables and configuring physical layer, next step is to create an NS object. In the simulations files, it is set and assigned to a variable "ns_" (see code 3.13) which is then used throughout the simulation for performing different operations.

```
01:   set ns_ [new Simulator]
```

Code 3.13: NS object creation

For wired cum wireless simulations, hierarchical routing is used for routing packets between wired and wireless domains. Packets are exchanged between wired and wireless nodes by the help of base-stations that work as gateways in-between both domains. Wired and wireless nodes are separated from each other by putting them in separate domains. Sub-domains are known as clusters. Sample code is given below.

```
01:   $ns_ node-config -addressType hierarchical
02:   AddrParams set domain_num_ 2
03:   lappend cluster_num 1 1
04:   AddrParams set cluster_num_ $cluster_num
05:   lappend eilastlevel 107 46
06:   AddrParams set nodes_num_ $eilastlevel
```

Code 3.14: Hierarchy of nodes

In the above code, in line 01, node object is configured to have Hierarchical address type. In line 02, numbers of domains in the topology are defined as 2, one for the wired nodes and one for wireless nodes. Line 03 defines the clusters in domains i.e. both domains contain 1 cluster each. Line 05 defines the number of nodes in each cluster.

After setting the hierarchy of nodes, tracing is setup. Since this simulation consists of both wired and wireless domains, therefore both traces will be generated and written in the same trace file. In the following code, variable "`$val(tracefile)`" is seen instead of trace file name. This variable contains the name of trace file e.g. "`sim_111_2h8c_agg.tr`". In code 3.15, line 01 defines the output trace file whereas line 02 will record all traces in the given trace file.

Sample code is as follows.

```
01:   set tracefd [open $val(tracefile) w]
02:   $ns_ trace-all $tracefd
```

Code 3.15: Tracing setup for simulation

Next lines in the simulation code create and setup topology and topography. Their required syntax is shown in code 3.16. Line 01 creates a new topography object "topo". It keeps track of mobile-nodes movement in the given boundary. In line 02 topography object is assigned the boundary by the help of x and y co-ordinates and the nodes will be placed in a flat-grid. In

line 04, n new object “God” is created which stands for “General Operations Director”. Total numbers of mobile nodes are passed as an argument to this God object. This God object stores information about total number of mobile nodes and table containing the shortest number of hops from one node to the other. A simulation contains only a single God object that is declared globally. Simulation code is shown below.

```
01:   set topo [new Topography]
02:   $topo load_flatgrid $val(x) $val(y)

03:   set val(totalnn) [expr $val(nn_wireless) + $val(nn_wired) +
$val(nn_bsnodes)]

04:   create-god $val(totalnn)
```

Code 3.16: Setting up Topology and Topography

3.3.7 Nodes Creation

Next step involves the creation of nodes. There are three types of nodes in simulations, which are created in the following order.

- Wired nodes.
- Gateway.
- Wireless nodes.

3.3.7.1 Wired Nodes

First of all wired nodes are created. Since there are 43 wired nodes therefore they are created by help of “for” loop (code 3.17, line 01). Wired node “node_wired(0)” is connected to all other wired nodes in the network resulting in the “Star topology”. This node is connected with the MANET’s gateway. Hierarchical addresses are assigned to these nodes i.e. “1.0.x” where value of “x” is unique for every node (line 02). This implies the IP-address of respective node.

```
                          #create wired nodes

01:   for {set i 0} {$i < $val(nn_wired)} {incr i} {
02:        set node_wired($i) [$ns_ node 1.0.$i]
03:        puts ("Wired_Node",$i,"created")
04:   }
```

Code 3.17: Wired Nodes creation

3.3.7.2 Gateway

Gateway node plays a very important role in MANET. A gateway node is a wireless node responsible for communication between wired nodes and wireless nodes network.

```
                        #create 1 base-station node

01:   set BS(0) [$ns_ node 0.0.54]
02:   $BS(0) random-motion 0
03:        set ra [ $BS(0) set ragent_]
04:                  $ra set debug_ 1
05:                  $ra set rt_log_interval_ 1000
06:                  $ra set log_to_file_ 1
07:                  $ra set local_repair_ $val(lrep)
08:                  $ra set llfeedback_ $val(llfb)
09:                  $ra set hello_jittering_ 1
10:                  $ra set rreq_gratuitous_ 0
11:                  $ra set wait_on_reboot_ 0
12:                  $ra set internet_gw_mode_ 1
13:                  $ra set expanding_ring_search_ $val(expring)
14:                  $ra set default_route_ $val(defrte)
15:                  $ra visitors 0.0.0 0.0.1 0.0.2 0.0.3 0.0.4 0.0.5
0.0.6 0.0.7 0.0.8 0.0.9 0.0.10 0.0.11 0.0.12 0.0.13 0.0.14 0.0.15 0.0.16
0.0.17 0.0.18 0.0.19 0.0.20 0.0.21 0.0.22 0.0.23 0.0.24 0.0.25 0.0.26
0.0.27 0.0.28 0.0.29 0.0.30 0.0.31 0.0.32 0.0.33 0.0.34 0.0.35 0.0.36
0.0.37 0.0.38 0.0.39 0.0.40 0.0.41 0.0.42 0.0.43 0.0.44 0.0.45 0.0.46
0.0.47 0.0.48 0.0.49 0.0.50 0.0.51 0.0.52 0.0.53 0.0.54 0.0.55 0.0.56
0.0.57 0.0.58 0.0.59 0.0.60 0.0.61 0.0.62 0.0.63 0.0.64 0.0.65 0.0.66
0.0.67 0.0.68 0.0.69 0.0.70 0.0.71 0.0.72 0.0.73 0.0.74 0.0.75 0.0.76
0.0.77 0.0.78 0.0.79 0.0.80 0.0.81 0.0.82 0.0.83 0.0.84 0.0.85 0.0.86
0.0.87 0.0.88 0.0.89 0.0.90 0.0.91 0.0.92 0.0.93 0.0.94 0.0.95 0.0.96
0.0.97 0.0.98 0.0.99
```

Code 3.18: Gateway node creation

As can be seen in the above code on line 1, the hierarchical address of the gateway is “0.0.54”. The hierarchical address for the wireless nodes is “0.0.x”. Unlike wired nodes, for wireless nodes it starts from “0”. Since the gateway is a wireless node, therefore it is assigned with the wireless nodes hierarchical addressing. Random motion of the gateway node is disabled (line 2).

Gateway nodes have some additional functionality as compared to normal wireless nodes like routing for wired network and information about the wireless network they are attached to. To create a gateway node, routing agent of the underlying router protocol has to be created and then attached to the wireless node. Here, routing agent of AODV-UU is created. Little description about routing agent properties is given below.

Routing agent property	Description
debug_	To print log messages to console.
rt_log_interval_	Intervals between logging AODV-UU routing table to routing table log-file.
log_to_file_	Creates a log file for events.
local_repair_	Use local repair for AODV-UU.
llfeedback_	Use link layer feedback for AODV-UU instead of HELLO messages.
hello_jittering_	Jittering for HELLO messages.
rreq_gratuitous	If set, gratuitous RREP packet is unicasted by intermediate nodes to destination node. This results in learning route back to the source node. This property enforces gratuitous flag on all RREQ messages.
wait_on_reboot_	Delay of 15-seconds on reboot.
internet_gw_mode_	Runs corresponding node as gateway node.
expanding_ring_search_	Expanding ring search used for RREQ messages.
default_route_	Default route for AODV-UU

Table 3.9: Routing Agent Properties Description

Values used for the routing agent properties in the simulation script are explained next in the following table.

Property	Value	Explanation
`debug_`	`1`	No output to console.
`rt_log_interval_`	`1000`	Configured in milliseconds. Logs routing table to log file after 1000 milliseconds.
`log_to_file_`	`1`	Logging to a general file is on.
`local_repair_`	`0`	Local repair for AODV-UU is off.
`llfeedback_`	`0`	Link layer feedback is off. Use HELLO messages.
`hello_jittering_`	`1`	Hello jittering is on.
`rreq_gratuitous_`	`0`	Set to off.
`wait_on_reboot_`	`0`	No delay on reboot.

internet_gw_mode_	1	Configures this node as gateway node.
expanding_ring_search_	1	Expanding ring search is used.
default_route_	0	No default route.

Table 3.10: Routing Agent Properties value with little explanation

As this node is connected with both wired and wireless nodes and route packets in between both networks therefore “wired routing” option needs to be enabled as can be seen in the following code.

```
$ns_ node-config -wiredRouting ON \
```

Code 3.19: Gateway node wired routing

If packet aggregation is used in the simulation an additional setting is also necessary to be done. In order to configure a node having packet aggregation, the queue type for that node is set to “ifqa”, which is the queue type of node having packet aggregation. In the simulation script it is done in the following manner.

```
$ns_ node-config -ifqType $val(ifqa) \
```

Code 3.20: Queue type for node having Packet Aggregation

Also for de-aggregation, a de-aggregator agent needs to be attached to the corresponding node. A numerical number “58” is written with the de-aggregator agent that corresponds to aggregated “IPMETA” packet. In the Tcl script, it is written in the following way.

```
$BS(0) attach [new Agent/Deaggregator] 58
```

Code 3.21: Attaching De-aggregator agent to node.

In case of an ordinary node, it is configured in the following way.

```
$ns_ node-config -ifqType $val(ifq) \
```

Code 3.22: Queue type for ordinary node.

3.3.7.3 Wireless Nodes

After creating gateway node, wireless nodes are created. In the simulation setup, 100 wireless nodes are created which is automated by the help of a “for” loop (code 3.23, line 01).

```
                        # create the wireless nodes

01:   for {set j 0} {$j < $val(nn_wireless)} {incr j} {
02:        set node_wireless($j) [ $ns_ node 0.0.$j ]
03:        $node_wireless($j) base-station [AddrParams addr2id \
                     [$BS(0) node-addr]]
04:        set position_x [expr ($j % 10) * 200 ]
05:        set position_y [expr ($j / 10) * 200 ]
06:        puts
("Wireless_Node",$j,":at_position:",$position_x,",",$position_y,"created")
07:        $node_wireless($j) set X_ $position_x
08:        $node_wireless($j) set Y_ $position_y
09:        $node_wireless($j) set Z_ 0.0

10:        set r [ $node_wireless($j) set ragent_]
11:           $r set local_repair_ $val(lrep)
12:           $r set llfeedback_ $val(llfb)
13:           $r set hello_jittering_ 1
14:           $r set rreq_gratuitous_ 0
15:           $r set wait_on_reboot_ 0
16:           $r set expanding_ring_search_ $val(expring)
17:           $r set default_route_ $val(defrte)
}
```

Code 3.23: Wireless nodes creation

These wireless nodes have the same network as of the gateway node i.e. “0.0.x” (line 02) and are given the same hierarchical address (line 03). Next, position for every node is calculated (lines 04-05) and set (lines 07-09). From lines 10-17, routing agent options for every node are set. They are same as of the gateway node except for “`internet_gw_mode_`”, since this only is applicable to gateway node.

In NS, every mobile node needs to be configured before it is created. These options are configured once in the Tcl script, applicable to all mobile nodes, created. These options are configured in the following way in NS simulation.

```
                        # configure global node config

01:   $ns_ node-config -adhocRouting $val(rp) \
02:                                -llType $val(ll) \
03:                                -macType $val(mac) \
04:                                -ifqType $val(ifq) \
05:                                -ifqLen $val(ifqlen) \
06:                                -antType $val(ant) \
07:                                -propType $val(prop) \
```

```
08:                                 -phyType $val(netif) \
09:                                 -channelType $val(chan) \
10:                                 -topoInstance $topo \
11:                                 -wiredRouting OFF \
12:                                 -agentTrace  $val(agttrace) \
13:                                 -routerTrace $val(rttrace) \
14:                                 -macTrace $val(mactrace) \
```

Code 3.24: Configuration for Wireless nodes

Small description with their configured values is explained in the next table.

Configuration Parameter	Configured Value	Description
-adhocRouting	AODVUU	Routing protocol used by the mobile node. In given case AODV-UU is used.
-llType	LL	Defines link layer type. For Satellite networks "LL/Sat" is used.
-macType	Mac/802_11	Mac 802.11 protocol is used.
-ifqType	Queue/DropTail/PriQueue	Defines interface queue type. Here "priority queue" is used.
-ifqLen	50	Maximum packets in IFQ.
-antType	Antenna/OmniAntenna	For wireless nodes, Omni-Antenna is used.
-propType	Propagation/TwoRayGround	Defines propagation model. Two ray ground reflection model is used.
-phyType	Phy/WirelessPhy	Physical layer for wireless networks is used.
-channelType	Channel/WirelessChannel	Wireless channel is configured. For Satellite networks, Channel/Sat will be used.
-topoInstance	$topo	The topography object is given in this field.
-wiredRouting	OFF	Wired routing is disabled for wireless nodes, since they will communicate only with wireless nodes.
-agentTrace	ON	Agent trace is enabled.

-routerTrace	OFF	Router trace disabled since they are not needed for this study.
-macTrace	OFF	MAC traces are also disabled.

Table 3.11: Wireless Node configuration parameters description

3.3.8 Connections between Nodes

Next step is of creating connections between nodes. General presentation of creating link between two nodes is described below.

```
$ns_ duplex-link <node1> <node2> <bw> <delay> <qtype> <args>
```

Code 3.25: General presentation of nodes connection

Explanation about these parameters for creating connection between nodes is as follows.

Parameter	Description
duplex-link	Defines link type. Duplex link is a bi-directional link.
<node1> <node2>	Defines nodes in between the connection is created.
<bw>	Defines bandwidth of the link in Mb.
<delay>	Defines delay of link in Ms.
<qtype>	Defines the queue type.
<args>	Additional arguments are passed by this parameter.

Table 3.12: Parameters explanation for creating connection

Simulations scenario is set in such a way that the gateway node "$BS(0)" is connected with only one wired node "$node_wired(0)". A bi-directional link "duplex-link" is created in between then so that they can send and receive network traffic. Link bandwidth is set to "5 Mb" having delay of "5 Ms" and queue type as "Drop-Tail".

```
                    #create link between wired node and bs
01:    $ns_ duplex-link $node_wired(0) $BS(0) 5Mb 5ms DropTail
```

Code 3.26: Connecting Gateway and Wired node

Next, wired node "`$node_wired(0)`" is then connected with all the other wired nodes by the help of a "for" loop as shown in code 3.27 at lines 01-02. Link type is again "`duplex-link`" having bandwidth "100 Mb", delay "0.2 Ms" and queue type as "`DropTail`".

```
                    #Create links between wired nodes

01:   for {set x 1} {$x < $val(nn_wired)} {incr x}
02:   $ns_ duplex-link $node_wired(0) $node_wired($x) 100Mb 0.2ms DropTail
```

Code 3.27: Connecting wired nodes

3.3.9 Setting Seed for Traffic Generation

In the next step, random variable seed is set. Each simulation is repeated 5 times with different seeds, hence "14 * 5 = 70" simulations were performed. Reason behind doing this was to get the confidence interval. Respective code is shown below.

```
01:   # set Random variable
02:       set rng  [new RNG]
03:       $rng seed 5
04:       $rng uniform 0 1
05:       set start_time [expr 0 + [$rng uniform 0.001 0.01]]
```

Code 3.28: Setting Random Seed

3.3.10 Setting Game Server

In the next step, game server is created and configured. The game server plays a very important role for game players. Quake3 server is placed in the wired network since it needs to be up and running all the time in order to facilitate game players who can join or leave game any time. The following code shows how a game server is created and configured.

```
                            #Quake3 Server

      #Create a UDP agent and attach it to node node_wired(1)
01:   set udp1 [new Agent/UDP]
02:   $udp1 set class_ 1
03:   $ns_ attach-agent $node_wired(1) $udp1
04:   set server1 [new Application/Traffic/Quake3Server]
05:   $server1 attach-agent $udp1
06:   $server1 set nrOfPlayers_ 9
      #Create a Null agent (a traffic sink)
07:   set null1 [new Agent/Null]
08:   $ns_ attach-agent $node_wired(1) $null1
```

```
09:   $ns_ connect $udp1 $null34
```

Code 3.29: Quake3 Server code

In order to create a game server, first of all a UDP agent "`udp1`" is created and attached to the desired wired node i.e. "`$node_wired(1)`" (code 3.28, lines 01-03). Then server instance is created (line 04) as "`server1`" and previously created UDP agent "`udp1`" is attached with this server object (line 05). After this, number of game clients are set that will connect with the game server, in this code, number of players are set to "9" (line 06). For receiving traffic a NULL agent is created (line 07) and then attached to the server node "`$node_wired(1)`" (line 08). In order to send traffic from game server to game client, UDP agent of game server must be connected to respective NULL agent of wireless node. In line 09, UDP agent of server "`$udp1`" is connected to the NULL agent of wireless node "`$null34`". Similarly, UDP agent of server must be connected to all other game client's NULL agents.

Till this point, game server is created in the fixed network with all necessary configuration and settings.

3.3.11 Setting Game Client

After creating game server, next step involves creation of game clients. Traffic is generated by game players that reaches game server. Game server sends update packets to its clients. Game clients are in the wireless network.

Game traffic from the game client first reaches at the gateway node which is then forwarded to the fixed node "`$node_wired(0)`" that is connected directly with the wireless gateway node. This fixed node then forwards the game traffic to game server node. Game server also sends update packets through the same channel. Code for Quake3 client is shown below.

```
                    #Quake3 Client Wireless Node 34

      #Create a UDP agent and attach it to node node_wireless(34)
01:   set udp34 [new Agent/UDP]
02:   $udp34 set class_ 2
03:   $ns_ attach-agent $node_wireless(34) $udp34
      #Create a Null agent (a traffic sink) and attach it to node
node_wireless(34)
04:   set null34 [new Agent/Null]
05:   $ns_ attach-agent $node_wireless(34) $null34
06:   set client34wireless [new Application/Traffic/Quake3Client]
07:   $client34wireless attach-agent $udp34
      #Connect the traffic source with the traffic sink
08:   $ns_ connect $udp34 $null1
```

```
                        #In case of Packet Aggregation
09:    Queue/Aggregator set nodeid_ 34
       #set dst for aggregation tunnel
10:    Queue/Aggregator set dstnodeid_ 54
11:    $node_wireless(34) attach [new Agent/Deaggregator] 58
```

Code 3.30: Quake3 Client code

Like Quake3 server, UDP agent is also created for game clients. In code 3.29, UDP agent "`udp34`" is created and then attached to the wireless node "`$node_wireless(34)`" (lines 01-03). Null agent (traffic sink) is created and attached to same wireless node for receiving traffic (lines 04-05). In line 06, game client's instance is created as "`client34wireless`". Previously created UDP agent "`udp34`" is then attached to this game client (line 07). After this, traffic source is connected with the traffic sink so that the traffic generated by the source node is received at the target node. In the above code, traffic source is the game client and the traffic target is game server. In line 08, UDP agent of the game client "`$udp34`" is connected with the NULL agent of the game server "`$null1`". Code from line 01 to line 08 is common for both categories of simulations i.e. aggregation and no aggregation.

If packet aggregation is implemented, game client needs to be configured with some additional settings relating to aggregation. Code lines from 09 to 11 consist of these settings. An aggregation tunnel is created in between game client and gateway node. Source of aggregation tunnel is the game client therefore Aggregation queue is set on the wireless game client (line 09). Destination of the aggregation tunnel is set to the gateway node therefore Aggregator queue's destination node is set to gateway node (line 10). For the aggregated packets coming from gateway node that are sent by the server also need to be de-aggregated, hence a de-aggregator agent is attached to the wireless game client "`$node_wireless(34)`" (line 11) where "58" represents numerical representation of IP_META packet.

Packet aggregation scheme is implemented in such a way that aggregation tunnels are created in-between game client wireless nodes and the gateway node. Game packets generated by the game client are aggregated into IP_META packets whose destination is set to the gateway node which is the end of aggregation tunnel. Packets after reaching at gateway, gets de-aggregated into original game packets and then send towards the game server. Similarly packets coming from the game server first reach at the gateway node where again aggregation tunnels are created according to their destination. IP_META packets are created at gateway node and sent towards the corresponding wireless game client. These packets after reaching at the game client are de-aggregated into QUAKE3P packets.

3.3.12 Scheduling Events

Last and important step includes scheduling events for all game server and clients. It consists of two steps.

- Step 1 includes “start” time for traffic agent.
- Step 2 includes the “stop” time of traffic agent.

Its syntax consists of three parts. Explanation of these parts is given below.

Required Syntax	Time	Event
`$ns_ at`	`$start_time`	`“server1 start"`
`$ns_ at`	`$val(runtime)`	`"$server1 stop"`

Table 3.13: Syntax for Scheduling Event

Time interval is mentioned in seconds. In table 3.13, time field consists of two values i.e. “`$start_time`” and “`$val(runtime)`”. “`$start_time`” gets starting time from random seed whereas “`$val(runtime)`” is the global variable that’s given a numerical value which is when used here is taken as in seconds. First event starts game server at “`$start_time`” seconds after the simulation starts and second event stops game server at the time stored in ““`$val(runtime)`”. Code for scheduling 8 game clients and 1 game server is mentioned in the following snippet.

```
                    #Schedule events for the agents

01:   $ns_ at $start_time "$server1 start"
02:   $ns_ at $val(runtime) "$server1 stop"
03:   $ns_ at $start_time "$client34wireless start"
04:   $ns_ at $val(runtime) "$client34wireless stop"
05:   $ns_ at $start_time "$client43wireless start"
06:   $ns_ at $val(runtime) "$client43wireless stop"
07:   $ns_ at $start_time "$client45wireless start"
08:   $ns_ at $val(runtime) "$client45wireless stop"
09:   $ns_ at $start_time "$client52wireless start"
10:   $ns_ at $val(runtime) "$client52wireless stop"
11:   $ns_ at $start_time "$client56wireless start"
12:   $ns_ at $val(runtime) "$client56wireless stop"
13:   $ns_ at $start_time "$client63wireless start"
14:   $ns_ at $val(runtime) "$client63wireless stop"
15:   $ns_ at $start_time "$client65wireless start"
16:   $ns_ at $val(runtime) "$client65wireless stop"
17:   $ns_ at $start_time "$client74wireless start"
18:   $ns_ at $val(runtime) "$client74wireless stop"
```

Code 3.31: Scheduling Events

3.3.13 Starting Simulation

For starting any NS simulation, following command is necessary. Simulation will start from 0 second time up-to the stop time specified in the simulation script.

```
                              #Starting simulation
01:   $ns_ run
```

Code 3.32: Starting Simulation

4 Simulations and Evaluation

In this chapter, I am going to explain the simulation setup employed to study the effect with and without packet aggregation on the network traffic of a popular First Person Shooter game "Quake3". Different simulations were carried out with varying game players and their hop distance from gateway node. Results were obtained from internet connected MANET having game traffic with two basic scenarios:

- With Packet Aggregation.
- Without Packet Aggregation.

4.1 General Simulation Setup

Simulation scenario is obtained from a previous study conducted in [33]. Basic scenario is common for every simulation. Mobile nodes in the wireless network are static. Total numbers of mobile nodes are 100 and wired nodes are 43. Mobile nodes are arranged in a 10 x 10 grid. Distance between any two mobile nodes is 200 meters resulting in a grid span of 2000 x 2000 meters.

Wireless network consists of only one gateway node. One of the mobile nodes from 100 nodes serves as gateway. All game clients belong from the wireless network whereas the game server is placed in the wired network.

Wireless network's data rate is 24 Mbps. Network connection between wired nodes is 100 Mbps with a delay of 0.2 Ms. Connection between the wired node and wireless gateway node is 5 Mb and delay of 5 Ms. Queue type used for all connection is "DropTail".

First node in the simulation scenario is QUAKE3 game server (colored black). All the fixed nodes are directly connected to node 0. Wireless gateway node is colored in green that is connected with node 0. Game clients are marked in red color.

Illustration of basic scenario is as follows.

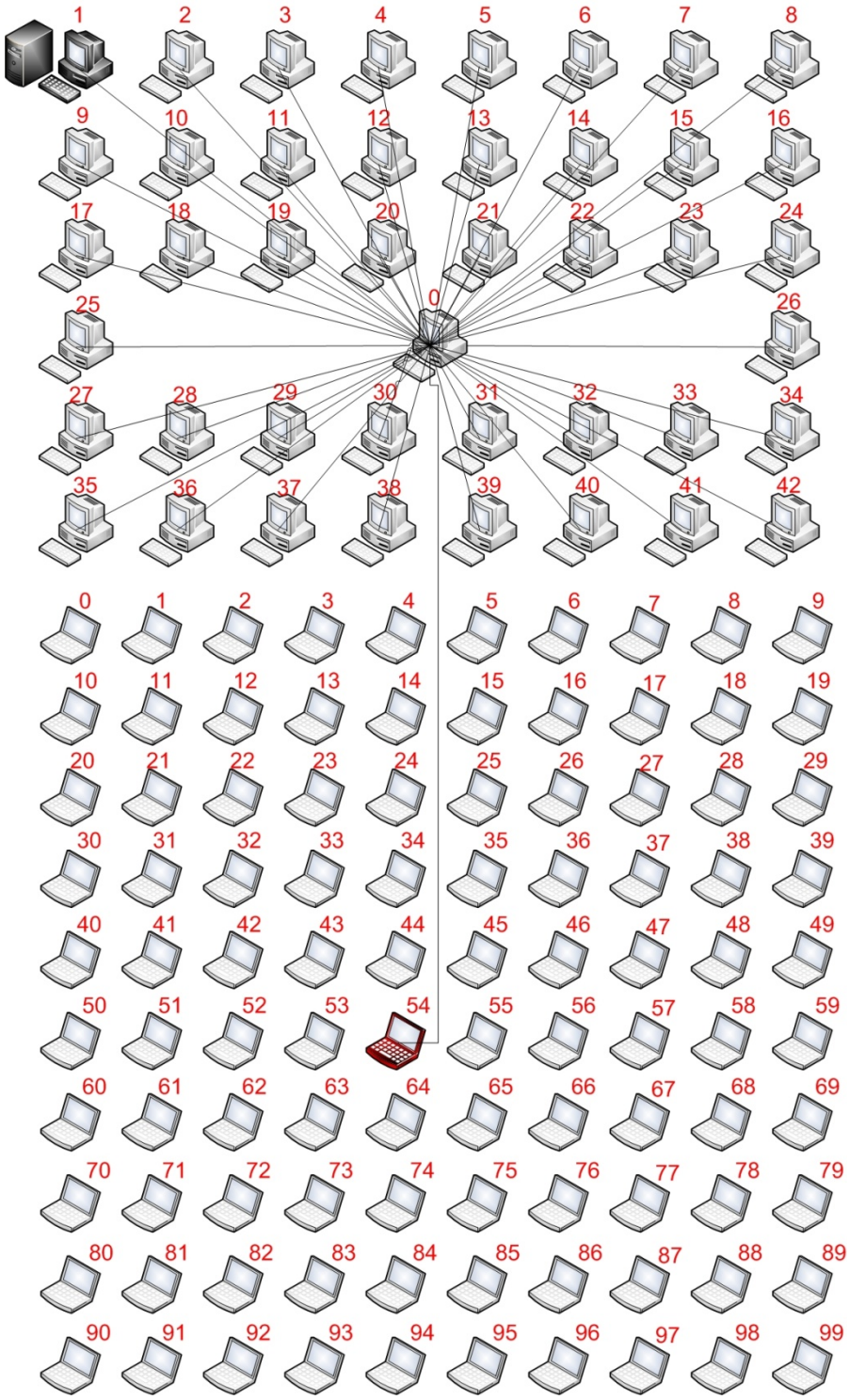


Figure 10: Basic Scenario

All nodes are numbered in NS. Some of the important nodes and their numbers are:

- Game Server = Node 1.
- Wireless Gateway = Node 54.
- Fixed node connected with game server and gateway = Node 0.

4.2 Game Traffic Scenario

This scenario consists of QUAKE3 game traffic flowing between both wired and wireless network. Simulations can be mainly categorized in 2 types. One with packet aggregation whereas the other without packet aggregation. They are further divided into 2 types. One with fixed clients and varying hops whereas the other with fixed hops and varying clients. A total of 14 different simulations were carried out for obtaining the end results.

This classification is shown in tabular form as follows.

With Packet Aggregation		**Without Packet Aggregation**	
Fixed Game Clients (8-Clients)	Fixed Hops (2-Hops)	Fixed Game Clients (8-Clients)	Fixed Hops (2-Hops)
2 hops	2 clients	2 hops	2 clients
3 hops	4 clients	3 hops	4 clients
4 hops	6 clients	4 hops	6 clients
5 hops		5 hops	

Table 4.1: Classification of Simulations for Game Traffic

General simulation settings common in all simulations are as follows. Description about these settings has been given before in previous chapter.

Settings	**Values**
MAC layer	`Mac/802_11`
Interface queue type	`Queue/DropTail/PriQueue`
Aggregator node queue type	`Queue/Aggregator`
Antenna model	`Antenna/OmniAntenna`
Interface queue length	`50`
Routing protocol	`AODVUU`
AODV-UU local repair	`0`
AODV-UU link layer feedback	`0`
RTS/CTS	`OFF`
MAC Layer Settings	
Basic rate	`24Mb`
Data rate	`24Mb`
CW min	`15`

CW max	1023	
Slot time	9us	
CCA time	3us	
SIFS	16us	
Preamble length	96	
PLCP header length	40	
PLCP data rate	6e6	
Short retry limit	7	(Used for “short” frames whose size is less than RTSThreshold).
Long retry limit	4	(Used for “long” frames whose size is greater than or equal to RTSThreshold).
Physical Layer Settings		
Propagation model	TwoRayGround	
Carrier Sense Threshold	5.53241e-12	
Receiving Threshold	1.296e-10	
Bandwidth	24Mb	
Transmitting power	0.1	
Frequency	2.4e+9	
System loss factor	1.0	
Collision Threshold	10.0	

Table 4.2: General Simulation Settings

Some other simulation settings necessary for the nodes implementing aggregation queue are as follows.

Settings	Values
Debug output	False
Aggregation followup	True
Aggregation Threshold	600
MTU size	1400
Maximum delay	10

Table 4.3: Aggregation Queue settings

4.3 Simulations Description

In this section, I will explain simulations carried out along with their description. As I have mentioned earlier that a total of 14 simulations were carried out with varying scenarios regarding game players and their hop distance from gateway node.

Simulations were carried out in pair and each simulation scenario is studied with and without packet aggregation/de-aggregation. Reason behind doing so was to calculate the performance difference between aggregation and no-aggregation. Certain factors like packet loss, delay and jitter is calculated.

Description about these simulations is as follows.

4.3.1 Simulations 1 and 2 (2-hops, 8-clients)

Simulations 1 and 2 consist of 8 wireless game clients at a distance of 2-hops away from wireless gateway node. Wired node 0 is directly connected with the game sever. Game clients are nodes 34, 43, 45, 52, 56, 63, 65 and 74 respectively. Refer to figure-3 for node numbers.

Simulation 1 implements packet aggregation and de-aggregation whereas simulation 2 is without aggregation. Nodes implementing these schemes are all 8 game clients and gateway node. In TCL script, queue type for these nodes is “ifqa” i.e. aggregator queue, while for other nodes it is set to “ifq”.

4.3.2 Simulations 3 and 4 (3-hops, 8-clients)

These simulations consist of 8 wireless QUAKE3 game players present at 3-hops distance from gateway. In these simulations, hops are increased for both simulations as compared to the previous simulations. Again, simulation 3 implements packet aggregation de-aggregation and simulation 4 does not implement these techniques. Game clients consist of nodes 24, 33, 35, 51, 57, 73, 75 and 84. Refer to figure-3 for node numbers.

4.3.3 Simulations 5 and 6 (4-hops, 8-clients)

Simulations 5 and 6 scenario is designed with 8 game clients present at 4 hops away from the wireless gateway node. Again hops are increased with respect to the previous simulations. Simulation 5 is with aggregation while simulation 6 is without. Game clients consist of nodes 14, 32, 36, 50, 58, 72, 76 and 94. Refer to figure-3 for node numbers.

4.3.4 Simulations 7 and 8 (5-hops, 8-clients)

These are the last simulations with increasing hops. They consist of 8 game clients that are 5 hops away from wireless gateway node. In these two simulations, simulation 7 is with

aggregation and simulation 8 is without. Game clients consist of nodes 4, 12, 37, 40, 59, 71, 86 and 93. Refer to figure-3 for node numbers.

4.3.5 Simulations 9 and 10 (2-clients, 2-hops)

From here onwards, simulations are fixed with 2 hops while game clients will vary. Simulation 9 is with packet aggregation whereas simulation 10 is without aggregation. In these simulations only 2 QUAKE3 game clients are present. Game clients consist of nodes 52 and 56. Refer to figure-3 for node numbers.

4.3.6 Simulations 11 and 12 (4-clients, 2-hops)

Packet aggregation is implemented in simulation 11 while simulation 12 is simple. In these simulations, 4 game clients are present at 2 hops away from internet connected gateway node. Game clients consist of nodes 34, 52, 56 and 74. Refer to figure-3 for node numbers.

4.3.7 Simulations 13 and 14 (6-clients, 2-hops)

These are the last simulations where simulation 13 employs packet aggregation and simulation 14 is with aggregation in order to study the effect of game traffic in the same scenario. In these simulations, 6 wireless game clients are present that are 2 hops away from gateway. Game clients consist of nodes 34, 43, 52, 56, 65 and 74. Refer to figure-3 for node numbers.

4.4 Traffic Scenario

There are three types of traffic running simultaneously between wireless and wired nodes in each simulation. These are namely:

1. Background Traffic Flows.
2. SIP calls traffic.
3. Quake3 game traffic.

These are described as follows.

4.4.1 Background Traffic Flows

Background traffic consists of exponential traffic flows comprising of VOIP traffic. Since these flows are running in the background, that’s why they are called as background traffic. They are used to create network load. In all simulations, 4 background traffic flows are created. Three of these flows are between wireless nodes, while one is in between one wired

node and wireless node respectively. Source and destination nodes for these flows are chosen randomly.

This traffic is just to create network load. Background traffic nodes in simulation code are as follows.

```
01:   set sender(0)  $node_wireless(21)
02:   set sender(1)  $node_wireless(14)
03:   set sender(2)  $node_wireless(35)
04:   set sender(3)  $node_wireless(94)

05:   set receiver(0)   $node_wireless(25)
06:   set receiver(1)   $node_wireless(19)
07:   set receiver(2)   $node_wireless(39)
08:   set receiver(3)   $node_wired(36)
```

Code 4.1: Back ground Traffic Nodes

4.4.2 SIP Calls Traffic

There are 8 SIP invitation calls in all simulations. 16 wireless and wired nodes are participating in SIP traffic i.e. 8 sender nodes and 8 receiver nodes. 14 nodes are from wireless network while 2 are from wired network. When simulation time equals to 20 seconds, these nodes start to initiate SIP calls. These invitation calls are repeated after every 5 seconds.

SIP calls nodes are as follows.

```
01:   set caller1  $node_wireless(45)
02:   set invited1   $node_wireless(63)
03:   set caller2  $node_wireless(34)
04:   set invited2   $node_wireless(64)
05:   set caller3  $node_wireless(53)
06:   set invited3   $node_wireless(74)
07:   set caller4  $node_wireless(33)
08:   set invited4   $node_wireless(55)
09:   set caller5  $node_wireless(52)
10:   set invited5   $node_wireless(44)
11:   set caller6  $node_wireless(65)
12:   set invited6   $node_wireless(34)
13:   set caller7  $node_wireless(66)
14:   set invited7   $node_wired(18)
15:   set caller8  $node_wireless(57)
16:   set invited8   $node_wired(31)
```

Code 4.2: SIP calls Nodes

4.4.3 Quake3 Game Traffic

My basic emphasize is on the Quake3 game traffic. Section 4.3 contains description of nodes placement for Quake3 game traffic. Each scenario has different nodes placements. Each simulation is studied with/without packet aggregation, which is also explained earlier.

Quake3 game traffic runs in parallel with exponential (background) traffic and SIP traffic. All simulations are carried out with one internet connected gateway node which routes traffic in between wired and wireless networks. Each game client and gateway node aggregates and de-aggregates packets.

4.5 Post Tracing Analysis

NS records all events that took place during simulation's running time in an output file known as the "trace file". These events are explained in 2nd chapter. Name of the trace file is mentioned in the simulation's TCL script. This file is then analyzed to extract useful information regarding that particular simulation.

All the simulations illustrated above have wired-cum-wireless scenario. An excerpt from a trace file for a wired-cum-wireless simulation from one of the simulation is shown below.

```
s 0.058885617 _96_ AGT  --- 145 QUAKE3P 65 [0 0 0 0] ------- [0.0.52:0
1.0.1:1 32 0]
r 0.060380950 _43_ AGT  --- 113 IPMETA 182 [cf 0 36 800] ------- [0.0.52:-1
0.0.54:58 32 0.0.54]
s 0.060380950 _43_ AGT  --- 113 encap 89 [0 36 35 800] ------- [0.0.52:-1
0.0.54:-1 32 0.0.53]
+ 0.060381 43 0 QUAKE3P 61 ------- 2 0.0.52.0 1.0.1.1 -1 112
- 0.060381 43 0 QUAKE3P 61 ------- 2 0.0.52.0 1.0.1.1 -1 112
```

Table 4.4: Wired-cum-Wireless Trace Format

Explanation of trace format for a wired-cum-wireless simulation is as follows.

Trace Value	Name	Explanation
s	Event type	Event occurred.
0.058885617	Time	Time at which the event took place.
_96_	Node ID	Node responsible for event.
AGT	Trace source	Can be MAC or RTR trace.
---	Reason (if any)	Indicates reason for event for error or drop.

145	Packet ID	Each packet has a unique ID.
QUAKE3P	Packet type	Type of the packet.
65	Packet size	Size of packet in bytes.
[0	Transmission time	MAC layer transmission time.
36	MAC source	Source node's MAC address.
35	MAC destination	Destination node's MAC address.
800]	Type	Type code. Each type has a code.
-------	Flags	
[0.0.52:0	IP source	Source node's IP address with port number.
1.0.1:1	IP destination	Destination node's IP address with port number.
32	TTL	Time to live of packet.
0.0.54]	Next hop	Next hop of the packet.

Table 4.5: Wired-cum-Wireless Trace Explanation

To extract meaningful information from a trace file, I have used Perl scripting. Certain performance parameters like packet loss, jitter, delay, 95 percentile etc were calculated for every simulation.

4.6 Performance Parameters

Following performance parameters for each scenario for both simulations i.e. with aggregation and without aggregation were calculated in order to differentiate the performance difference for QUAKE3 game traffic.

- Average delay.
- Packet loss ratio.
- Jitter.
- Packets lost.
- 95 percentile.

These are explained below.

4.6.1 Average Delay

Delay is the time difference required in processing a packet from sender node to receiver node. This time difference is known as "end-to-end one-way delay" or latency [36]. End-to-

end delay consists of several different delays like at sender node, at network level and at receiver node.

At the sender node, various factors add delay e.g. processes like encoding, packetizing and serializing [37]. On the network level, packet is queued in different routers, forwarded to different nodes etc. all these factors increase the delay. On the receiver node, corresponding packet is de-packetized and de-coded which also adds to end-to-end delay. Game players prefer Quake3 severs having latency less than 150-180 ms [38].

In order to measure average delay, I need to calculate the time difference of each packet from its sending time to receiving time. This refers to "*packetDuration*".

$$\boldsymbol{packetDuration = endTime - startTime}$$

Equation 1: Packet Duration

This "packetDuration" is then added up to calculate the total time. Number of packets for which this time is calculated is also stored. Average delay is calculated by dividing packet duration with previously stored number of packets. It is shown in the next equation.

$$\boldsymbol{averageDelay = \frac{\sum packetDuration}{numberOfPackets}}$$

Equation 2: Average Delay

4.6.2 Packet Loss Ratio

A packet is considered lost when it is sent by the source node and is unable to reach the destination node because of getting corrupted or dropped. Causes of packet loss are congested network, signal strength, mobility, transmission errors and channel noise.

Packet loss ratio is calculated by the following equation.

$$\boldsymbol{packetLossRatio = 1 - (\frac{totalPacketsReceived}{totalPacketsSent})}$$

Equation 3: Packet Loss Ratio

4.6.3 Jitter

Jitter is known as the variation in time between arriving packets caused by network congestion, route changes or time drift [39]. Jitter is considered as the variation in delay of consecutive packets.

Jitter is calculated by the following equation of "standardDeviation". Standard Deviation of a probability distribution is defined as the square root of variance [42].

$$\boldsymbol{standardDeviation} = \sqrt{\frac{\mathbf{1}}{\boldsymbol{N}-\mathbf{1}}\sum_{\boldsymbol{i}=\mathbf{1}}^{\boldsymbol{N}}(\boldsymbol{X_i}-\bar{\boldsymbol{X}})^{\mathbf{2}}} \quad \textbf{[48]}$$

Equation 4: Standard Deviation (Jitter)

Standard Deviation is calculated by carrying out following steps [41].

1. For each value$\boldsymbol{X_i}$ (packet's delay), calculate the difference $\boldsymbol{X_i}-\bar{\boldsymbol{X}}$ between $\boldsymbol{X_i}$ and the average value$\bar{\boldsymbol{X}}$.
2. Calculates the squares of these differences.
3. Find the average of squared differences. This quantity is the variance.
4. Take the square root of the variance.

The end value obtained is the Standard Deviation.

4.6.4 Packets Lost

Packets lost are calculated by the help of following equation.

$$\boldsymbol{packetsLost} = \boldsymbol{packetsSent} - \boldsymbol{packetsReceived}$$

Equation 5: Packets Lost

4.6.5 95 Percentile of Delay

95 percentile rule states that, it is the smallest number, that is greater than 95% of numbers contained in the given set [40]. This is a mathematical calculation that helps to calculate accurate amount of delay. The 95th percentile says that 95% of time, delay is below this amount while remaining 5% of time, delay is above that amount.

Standard Error (stddev_error) is employed to calculate confidence limits for true population mean. Standard error is the estimated "standard deviation" of the error in that method [43]. It is the standard deviation of difference between measured values and true values.

Equation for calculating Standard Error is as follows.

$$\boldsymbol{stddevError = \frac{standardDeviation}{\sqrt{numberOfPackets}}} \quad \textbf{[48]}$$

Equation 6: Standard Error

For 95 percentile, two confidence limits are calculated namely Upper Confidence Limit (UCL) and Lower Confidence Limit (LCL). These limits are calculated by the following equations.

$$\boldsymbol{Upper\ 95\%\ Limit = averageDelay + (1.96 * stddevError)} \quad \textbf{[48]}$$

Equation 7: Upper 95% Limit

$$\boldsymbol{Lower\ 95\%\ Limit = averageDelay - (1.96 * stddevError)} \quad \textbf{[48]}$$

Equation 8: Lower 95% Limit

4.7 Simulations Results and Evaluation

This part of thesis report contains results from simulations carried out and their evaluation. Simulations topology and network traffic has been explained in earlier sections. A total of 14 simulations were carried out with varying game-clients and hops. Each simulation is repeated 5 times with different seeds to get the confidence interval.

For evaluating Quake3's game quality, two performance related key things needs to be kept in mind, which are as follows:

- Latency tolerance for player of Quake3 was empirically established to be between 150 – 180 milliseconds [38].
- Quake3 players are reasonably happy even up to 35% packet loss rate [44].

Above mentioned points tell about the upper bounds of Quake3 game traffic till which the game is playable i.e. delay is tolerable up till 180 milliseconds and packet loss is acceptable up till 35%.

4.7.1 Fixed Game Clients with Varying Hops

First set of simulations were carried out with fixed number of game clients placed at varying hops from gateway node in order to evaluate the performance difference of game traffic measured at each hop of game client over the wireless network.

Following table contains results obtained from different simulations without packet aggregation having fixed number of clients with varying hops. Since each simulation was repeated 5 times with different seeds, therefore "median" values are shown in the following tables.

Simulations	Packet Loss Ratio	Average Delay	Jitter
2-hops 8-clients	0.344891799	0.453000742	0.437687826
3-hops 8-clients	0.646618278	0.82945664	0.530350445
4-hops 8-clients	0.702331411	0.945463886	0.620545858
5-hops 8-clients	0.735582605	1.099864506	0.723406857

Table 4.6: Simulations Results without Packet Aggregation for Varying Hops

Whereas the next table contains simulations results carried out with packet aggregation bearing varying hops with fixed number of game clients.

Simulations	Packet Loss Ratio	Average Delay	Jitter
2-hops 8-clients	0.075816237	0.024711564	0.058950244
3-hops 8-clients	0.071005061	0.033104395	0.06655875

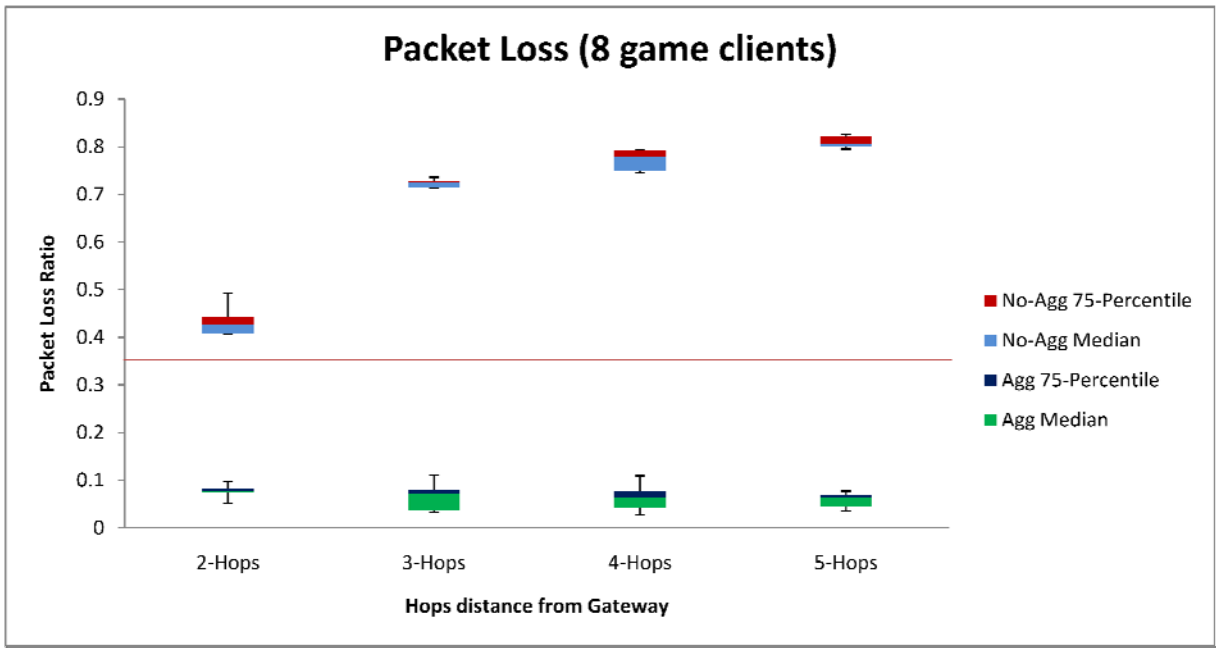
4-hops 8-clients	0.062900211	0.051089178	0.111364334
5-hops 8-clients	0.062733868	0.11348557	0.161585856

Table 4.7: Simulations Results with Packet Aggregation for Varying Hops

Performance difference between the two schemes can be seen by the following graphs.

4.7.1.1 Packet Loss Ratio

First graph shows the difference between "**Packet loss ratio**" for the same scenario for packet aggregation and without aggregation having 8 game clients placed at 2, 3, 4 and 5 hops away from wireless gateway node.



Graph 1: Packet loss for 8-clients with varying hops

Red line in the above graph indicates the upper limit for Quake3 packet loss i.e. 35% up till which the game is playable [44].

According to the above graph, packet loss of game traffic without packet aggregation is much higher than the case having packet aggregation. Scenario having no packet aggregation, packet loss percentage increases with the increasing number of hops whereas with packet aggregation, this ratio decreases with increasing hops and remains below 10%. As far as the packet loss for no-aggregation scenario is concerned, even for 2-hops it is higher than 40%, at

which according to [44], game is non-playable due to large packet loss. Therefore, from the above graph, it is concluded that Quake3 game is non-playable in simple scenario i.e. without packet aggregation.

Wireless network consists of game, SIP and background traffic. Since these packets are small in size and MAC has its own additional overheads i.e. SIFS, DIFS and contention which takes place for each packet, therefore network congestion increases. This increases the chance of collisions and high packet loss. When several small packets are aggregated into one large "IPMETA" packet, SIFS, DIFS and MAC contention occurs once for that larger IPMETA packet resulting in reduced network congestion and less packet loss ratio.

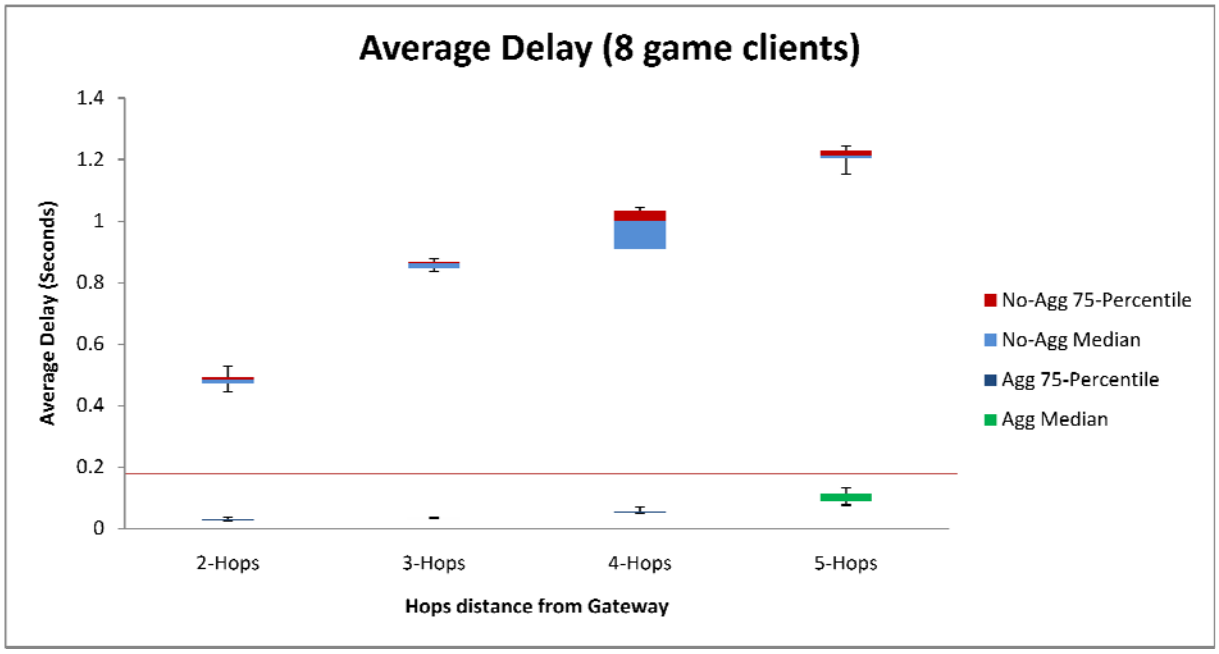
When hops are increased between game clients and gateway node, game traffic has to traverse through additional intermediate nodes. Additional nodes have to process the game traffic along with other network traffic i.e. SIP, exp etc. High network traffic increases network congestion. This can result in collisions ultimately increasing packet loss ratio.

Hence from the above graph it is proved that packet aggregation has significantly improved the packet loss ratio in the given wireless network scenario and keeps packet loss well within limits even for increasing hops (simulated successfully up till 5-hops) which in no-aggregation scenario is non-playable.

Note: *Reason for the high packet loss for packet aggregation was a bug in an earlier version of the aggregation algorithm combined with routing problems. To solve the routing problems with the aggregation modules is outside the scope of this thesis study.*

4.7.1.2 Average Delay

Next graph shows the difference of "**Average Delay**" between both scenarios with fixed game clients and varying hops.



Graph 2: Average Delay for 8-clients with varying hops

Red line in the above graph indicates the upper limit for Quake3 packets delay i.e. 180 milliseconds, up till which the game is playable [38].

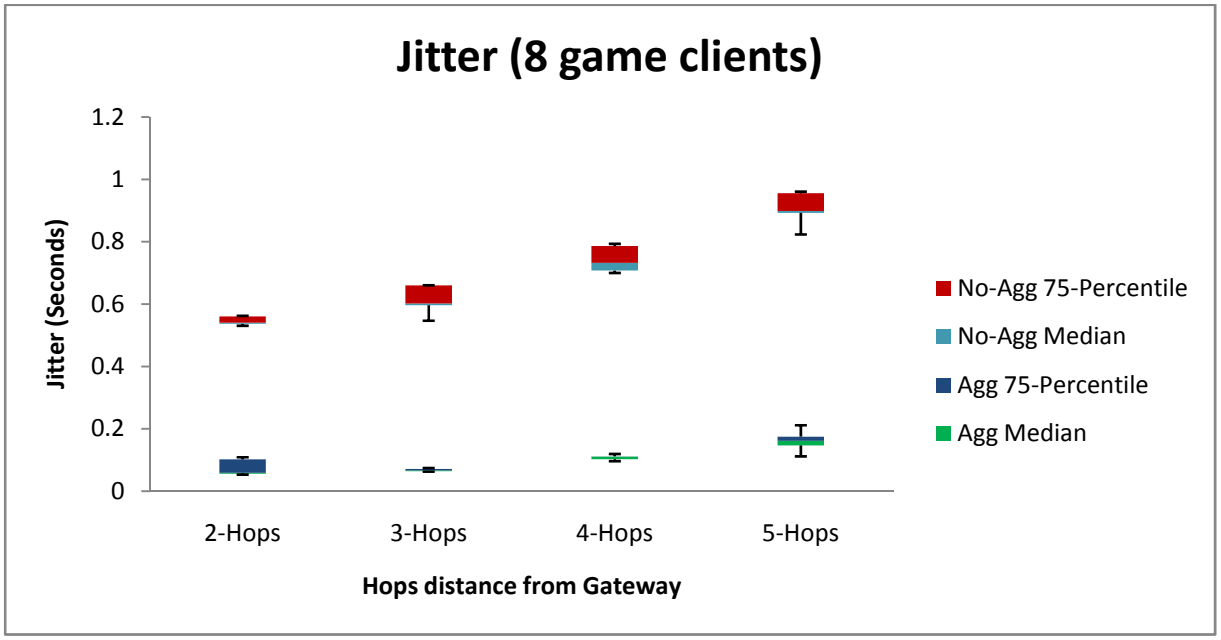
Above graph shows that average delay increases in both scenarios by increasing hops, but in no-aggregation scenario, it increases at a tremendous rate whereas in packet aggregation scenario, it increases gradually and remains below the value of 112 ms for 5-hops. Since the upper boundary for Quake3 packet delay is 180 milliseconds [38], packet aggregation keeps the game playable well within the limit, whereas without aggregation delay starts from 450 milliseconds for 2-hops which is too much for game players to play. At this high delay, game is not playable.

When hops are increased, game packets need to traverse through different intermediate nodes before reaching the gateway node. At each node, these packets are first received, queued and then sent to the next hop along with other network traffic. Each step adds additional delay to these packets. Higher numbers of packets increase overall end-to-end delay. Also this delay is increased by each additional hop. When these small packets are aggregated into larger IPMETA packet, this delay is reduced a lot and occurs only once for IPMETA packet that comprises of several small packets resulting in reduced end-to-end packet delay.

Hence packet aggregation has significantly improved the packets delay in the given wireless network scenario and keeps game playable for increasing hops (simulated successfully up till 5-hops) which in no-aggregation scenario is non-playable.

4.7.1.3 Jitter

"**Jitter**" of both scenarios is shown in the next graph having 8 game clients at each hop.



Graph 3: Jitter for 8-clients with varying hops

The above graph also shows a significant difference for Jitter between packet aggregation and without aggregation. Jitter increases with increasing hops in both cases but its percentage is much higher in no-aggregation scenario than packet aggregation scenario. Maximum jitter recorded for 5 hops in packet aggregation case is near 160 ms whereas in no-aggregation scenario it is more than 720 ms, which is 4.5 times higher.

4.7.2 Fixed Hops with Varying Game Clients

The next sets of simulations were carried out with fixed number of hops while changing number of game clients in wireless network in order to evaluate the affect of increasing game traffic. Game clients, in each simulation, are 2-hops away from wireless gateway node. Number of game clients are gradually increased i.e. 2, 4, 6 and 8 as can be seen in the next tables.

Following table contains results obtained from different simulations without packet aggregation having fixed number of hops with varying game clients. Since each simulation was repeated 5 times with different seeds, therefore "median" values are shown in the following tables.

Simulations	Packet Loss Ratio	Average Delay	Jitter
2- clients 2- hops	0.001410189	0.022381936	0.060412118
4- clients 2- hops	0.014147179	0.05103743	0.115032762
6- clients 2- hops	0.146453574	0.244382522	0.301151559
8- clients 2- hops	0.344891799	0.453000742	0.437687826

Table 4.8: Simulations Results without Packet Aggregation for Varying Game Clients

The next table contains simulations results carried out with packet aggregation bearing varying game clients with fixed number of hops.

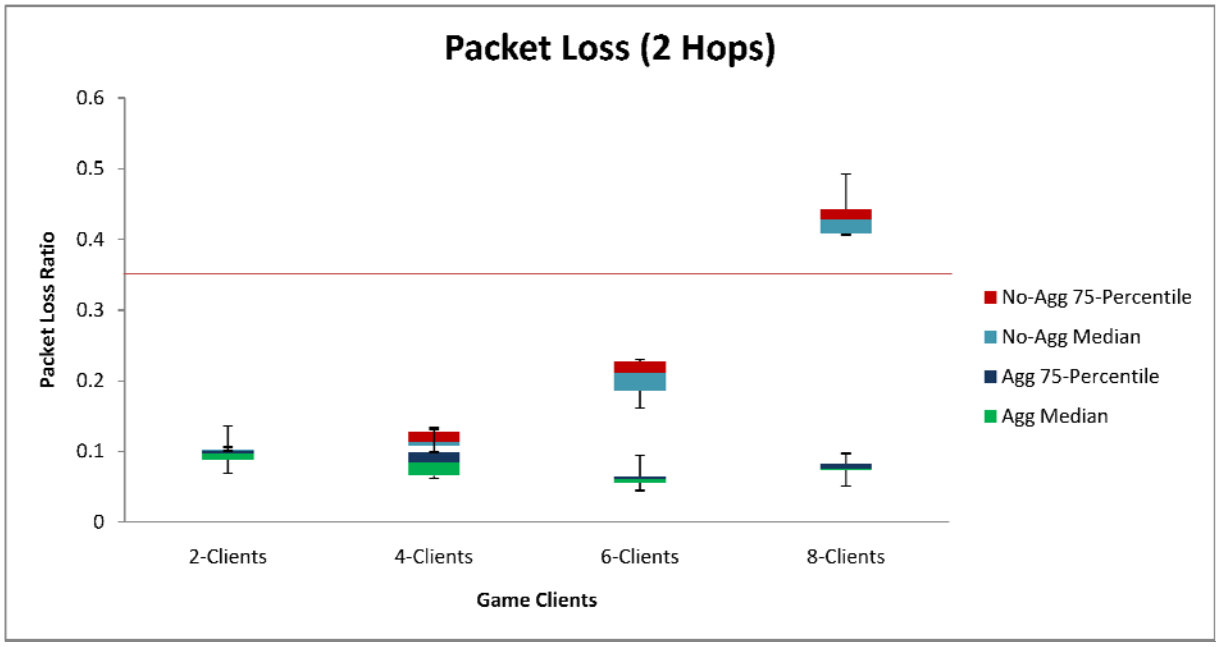
Simulations	Packet Loss Ratio	Average Delay	Jitter
2- clients 2- hops	0.096974985	0.032065501	0.05693689
4- clients 2- hops	0.085033275	0.028648554	0.055942929
6- clients 2- hops	0.060469301	0.028238733	0.061224876
8- clients 2- hops	0.075816237	0.024711564	0.058950244

Table 4.9: Simulations Results with Packet Aggregation for Varying Game Clients

Performance difference between both schemes can be seen in the following graphs.

4.7.2.1 Packet Loss Ratio

Following graph shows the difference of "**Packet Loss Ratio**" between packet aggregation and no-aggregation schemes.



Graph 4: Packet loss for 2-hops with varying game clients

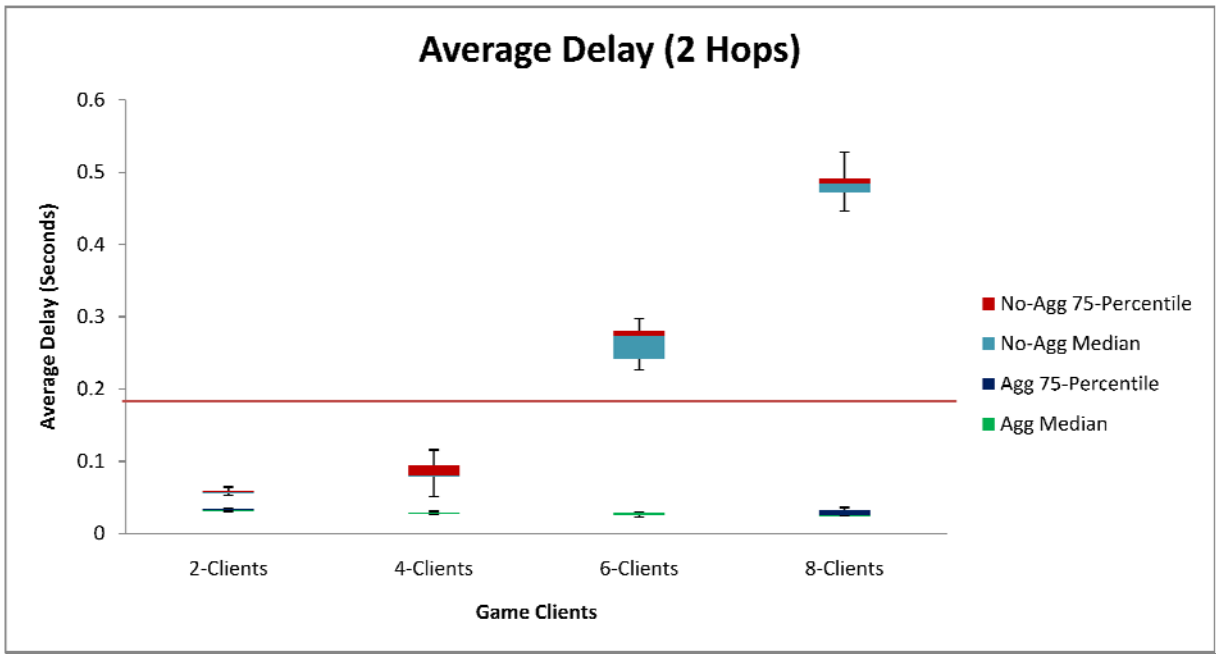
Here again in this graph, red line indicates the upper limit for Quake3 packet loss i.e. 35%. The above graph shows that when no packet aggregation is used, packet loss for 2 game clients is above than 10% and it keeps on rising when more game clients join the game. The game is playable up till 6 game clients present at 2-hops away from the wireless gateway node. For 8 game clients, game is not playable because of high packet loss exceeding the upper bound. Whereas in packet aggregation scenario, packet loss ratio for starting simulation of 2 game clients is about 10% which decreases with the increase in number of game clients. This indicates that packet aggregation provides better performance when network traffic is high. Even for 8 game clients, packet loss ratio is still lower than that of 2 game clients i.e. less than 10%.

Hence above graph proves that packet aggregation has less packet loss ratio as compared to no-aggregation scenario which keeps the game playable even for more game clients which is impossible in no-aggregation case (simulated successfully up till 8-game clients).

Note: *Reason for the high packet loss for packet aggregation was a bug in an earlier version of the aggregation algorithm combined with routing problems. To solve the routing problems with the aggregation modules is outside the scope of this thesis study.*

4.7.2.2 Average Delay

The following graph depicts difference in "**Average Delay**" for 2-hops.



Graph 5: Average Delay for 2-hops with varying game clients

Red line in the above graph indicates the upper limit for Quake3 packets delay i.e. 180 milliseconds, up till which the game is playable [38].

The above graph shows that average delay in no-aggregation scenario increases tremendously when more game clients join the game. This graph's output is almost similar to the previous ones. Game is playable up till 4 game clients. On the other hand, increase in number of game clients does not affect packet-aggregation scheme. It keeps on decreasing with the addition of more game players and remains playable even for 8 game clients.

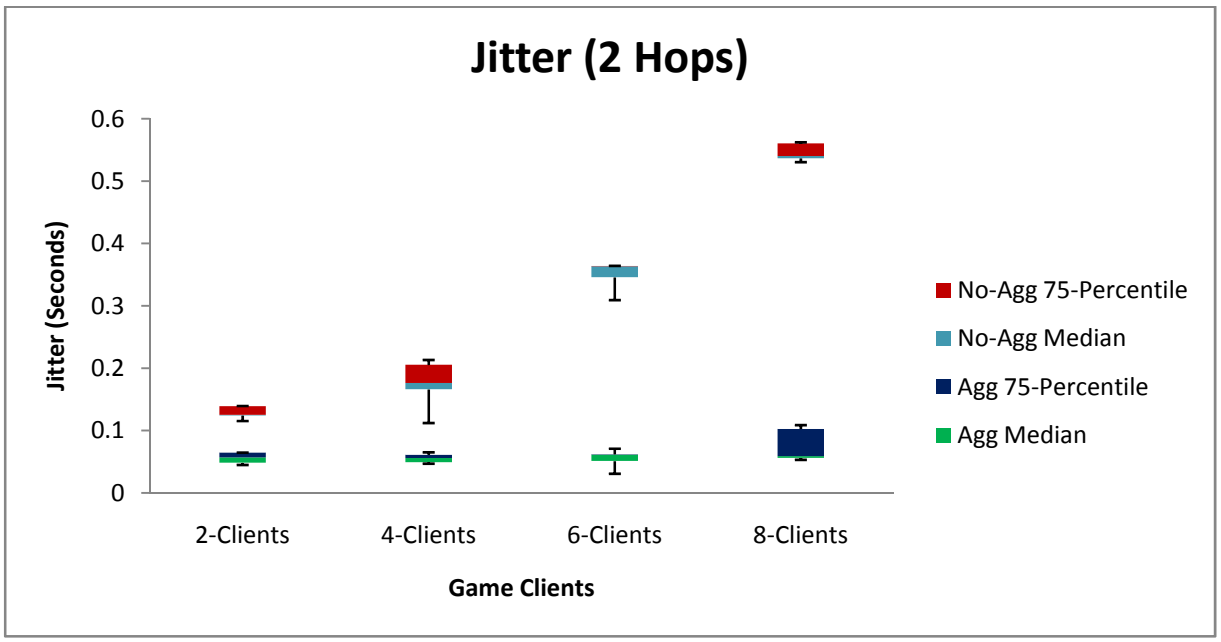
In packet aggregation case, average delay for 2-clients is about 32 milliseconds and it keeps on decreasing by increasing game clients and for 8-clients it becomes about 25 milliseconds. This indicates that packet aggregation shows better results when network traffic is high.

Average delay is high in no-aggregation scheme because each packet has its own MAC overhead i.e. SIFS, DIFS and contention window. Each of them introduces additional delay which adds to the overall end-to-end delay. But when different small packets are aggregated into large IPMETA packet, this overhead is reduced a lot since it takes place only once for IPMETA packet. That's why delay in packet aggregation scheme is less than no-aggregation scheme.

Hence the upper graph shows that average delay is less when packet aggregation scheme is used irrespective of increasing game clients and more players can play the game, which is not possible in no-aggregation case.

4.7.2.3 Jitter

The next graph explains "**Jitter**" difference between packet-aggregation and no-aggregation schemes.



Graph 6: Jitter for 2-hops with varying game clients

The above graph looks quite similar to the delay graph explained earlier. Jitter increases in both cases. In without aggregation case, it increases too much each time more game clients join the game whereas in aggregation case its magnitude is not too much and remains below 100 milliseconds even for 8 game clients.

Hence from the upper graph it is proved that packet aggregation scheme results in reduced jitter whereas no-aggregation scheme yields higher jitter.

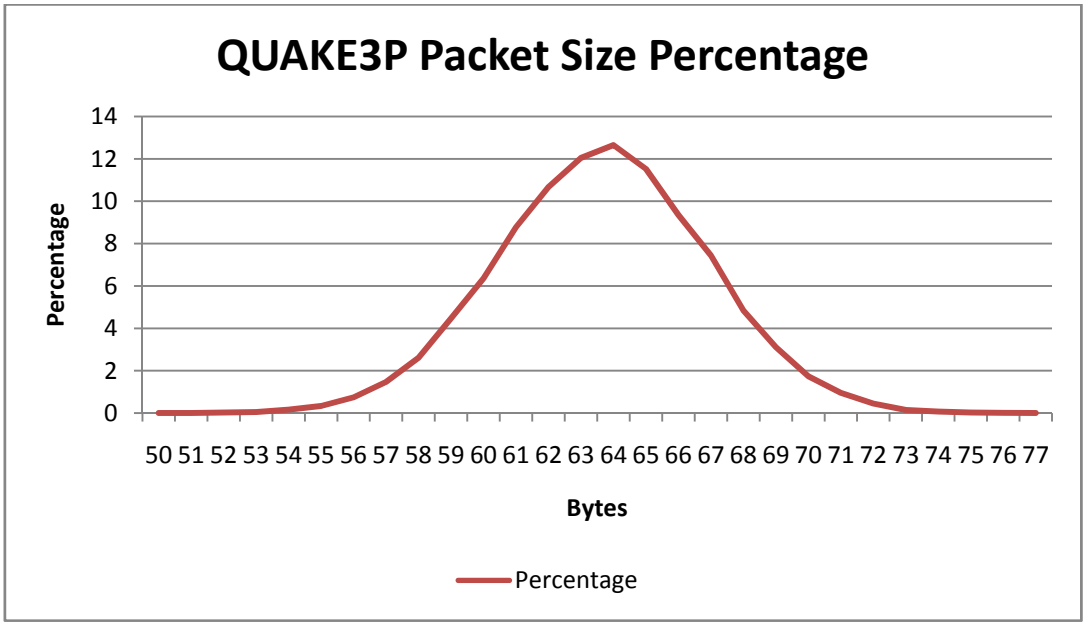
4.7.3 Packet Lengths

In this section packet lengths for simple game packets "QUAKE3P" and aggregated packets "IPMETA" in terms of bytes is discussed.

4.7.3.1 Simple Game Packets

Packet lengths of game traffic without any aggregation have a limited range between 58 to 70 bytes. This length does not depend on map played or number of game players, hence remains same for each game client playing the game.

Packet lengths with respect to percentage are shown in the following graph.

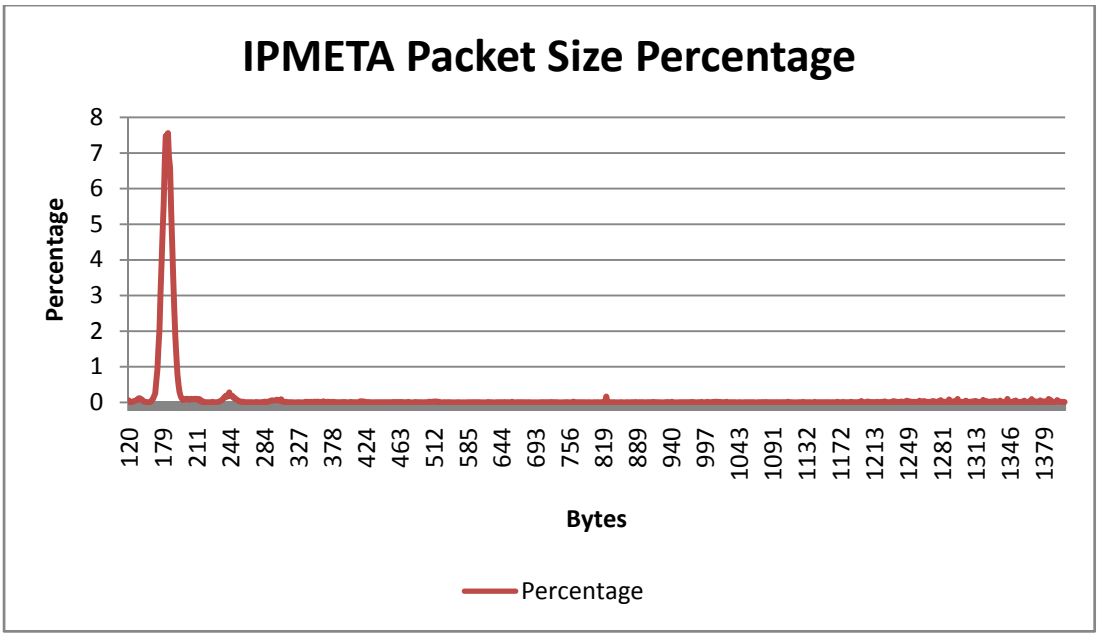


Graph 7: Packet size percentage of QUAKE3P packets

4.7.3.2 Aggregated Game Packets

An aggregated IPMETA packet consists of one or several QUAKE3P packets. Majority of aggregated IPMETA packet size lies between 170 bytes and 200 bytes. The minimum size of IPMETA packet is 120 bytes whereas the maximum size is 1399 bytes.

Packet size with respect to percentage is shown in the following graph.



Graph 8: Packet size percentage of IPMETA packets

4.7.4 Evaluation of Game Quality

Quake3 is a multiplayer internet based highly interactive First Person Shooter game in which players have relatively short reaction times. This characteristic makes this game sensitive to network characteristics [38].

In order to analyze game quality in MANETs, performance parameters needs to be evaluated i.e. packet loss ratio, average delay etc. These parameters have been evaluated from obtained results in the previous section.

In [38] the author has studied the latency sensitivity among players and has concluded that game players prefer servers having latency less than 150 to 180 milliseconds from player's location. If it is greater than this value then it will result in greater delays that can lead to non-playable game. If we see on the "Average Delay Graph (4.7.1.2)", delay for 2 hops and 8 game clients is more than 450 milliseconds and it keeps on increasing with increasing hops, in a normal scenario i.e. without packet aggregation, which is too high for a highly interactive game like Quake3. This is so because an additional delay is added by every intermediate node since there are queuing delays, processing delays and MAC contention. When hops are increased, delay is also increased. If packet aggregation is applied to the same scenario, it improves the average delay dramatically and even for 5-hops, delay remains below 115 milliseconds which is well under the limit of 180 milliseconds. According to this graph, due to high packet delay, game is non-playable in the simple scenario i.e. without aggregation even for 2 hops whereas in aggregation case, game players can play the game without any

problem (simulated successfully up till 5 hops). Also, if we see on “Average Delay Graph (4.7.2.2)”, delay for 2-hops with no-aggregation is higher than with aggregation scheme. Here again packet aggregation is dominant over no-aggregation with the reasons mentioned earlier and is near 25 milliseconds for 8 game clients. Game remains playable at most for 4 game clients in no-aggregation case and in aggregation case, more game clients can play the game without too much delay (simulated successfully up till 8-game clients). Hence packet aggregation improves network delay to a great extent, in favor of game players.

Packet loss study for Quake3 had been carried out in [44]. The authors have showed that Quake3 shows no quality degradation for small packet loss rates i.e. till 35%. Quake3 players, even at higher delays, perceive the game being responsive. Players are reasonably happy even up to 35% packet loss rate. This perceived performance is achieved by the client-side prediction algorithm that is running around to work out for the missing information caused by packet loss. Quake3 is explicitly designed to “hide” latency (command executed at client-side) that immediately shows a result on screen by help of local prediction. The authors also have proved that QoS of Quake3 is also not degraded by the prediction algorithm. The number of deaths and kills is almost unaffected by packet loss. Difference of packet loss between packet aggregation and no-aggregation is shown in the previous section i.e. “4.7.1.1” and “4.7.2.1”. In “4.7.1.1”, packet loss in no-aggregation scheme for 2-hops is more than 40% and it keeps on increasing with increasing hops. For 5-hops it exceeds 75%, which is too much for a game like Quake3 that demands quick response. But results are quite different when packet aggregation is applied to same scenario. Even for 5-hops, packet loss remains under 10%, which is well under the desirable packet loss limit (35%). If we see the graph, packet loss for 2-hops was under 10% and remained under 10% even for 5-hops. According to this graph, in no-aggregation case, Quake3 game is non-playable even for 2 hops whereas in aggregation case, game is playable for increasing hops also (simulated successfully up till 5 hops). The next graph “4.7.2.1” also shows the difference between both schemes where all the game clients are at 2-hops away from wireless gateway node. Here again packet aggregation has shown better results for increasing number of game clients and remains under 8% for 8 game clients. As far as no-aggregation scheme is concerned, packet loss for 6 game clients is under the upper limit. For more than 6 game clients, game becomes non-playable.

Therefore, from the obtained results, it is proved that packet aggregation in both scenarios has performed tremendously well and showed better results as compared to no-aggregation scheme and keeps game playable even in the scenarios when it is impossible to play without packet aggregation.

5 Conclusion and Future Work

In this thesis, I have researched the performance of an interactive online first person shooter game Quake3 with respect to packet aggregation and no-aggregation in hybrid MANET. In recent years, interactive network games have become very popular and game traffic growth is prevailing on the internet. Interactive games have strict QoS requirements than other web or email traffic. When such games are played over wireless network, their traffic is more prone to fluctuations, packet loss and delay that can further degrade the game quality. In a hybrid MANET, wireless nodes send their traffic out of MANET by the help of gateway nodes. Hence the game traffic needs to pass through gateway twice i.e. outgoing and incoming, which further limits the performance.

Quake3 game traffic consists of small packets. Wireless network is prone to fluctuations and has no stable bandwidth. These small packets create congestion in the network which increases packet loss and delay hence result in degraded game quality.

The technique used in this paper to improve game quality in MANET is packet aggregation [3]. Difference between the performance of scenarios with packet aggregation and no-aggregation is evaluated by the help of several simulations that were carried out in NS simulator version 2.26. The scalability is evaluated with respect to varying number of hops and game clients in each simulation.

Depending on the results collected from simulations, it is concluded that packet aggregation showed great performance related results as compared to no-aggregation with respect to packet loss, delay and jitter.

With packet aggregation, game remains playable well under the upper bounds of quality i.e. delay must be less than 150 - 180 milliseconds and packet loss up till 35% which is not possible without aggregation.

Hence packet aggregation improves game quality a lot in favor of players over mobile ad-hoc wireless network and keeps it playable for greater hops and more game clients.

Although packet aggregation scheme has shown tremendous results but in future, this game traffic can be studied with the updated packet aggregation algorithm that is free of bugs that were present in the version used in this thesis study in order to get more updated results. That updated algorithm is adaptive to the network conditions i.e. does not aggregate packets when network traffic is low and aggregate when traffic becomes high. Also, one more thing can be done is to check this packet aggregation scheme in a real scenario i.e. real nodes and game clients with packet aggregation implemented on them to verify the obtained results for

game performance and quality. Those results will tell in real the game performance over Mobile Adhoc NETwork.

References

[1] S. McCreary, K. Claffy. *Trends in Wide Area IP Traffic Patterns: A View from Ames Internet Exchange.* 13th ITC Specialist Seminar on *Measurement and Modeling of IP Traffic*, Sep 2000.

[2] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool. *The Effects of Loss and Latency on User Performance in Unreal Tournament 2003.* In Proceedings of *3rd Workshop on Network and System Support for Games*, Aug. 2004, pp. 144–151.

[3] Peter Dely. *Adaptive aggregation of VOIP in wireless mesh networks.* Master's thesis Karlstad University 2007.

[4] T. Lang, P. Branch and G. Armitage. *A synthetic traffic model for Quake3*. ACM SIGCHI ACE2004 conference, Singapore, June 2004.

[5] K. Fall and K. Varadhan, Editors. *The VINT Project: The ns Manual*. UC Berkeley, LBL, USC/ISI and Xerox PARC, 2007.

[6] Scott, M.C., J.P. Macket and G.H. Cirincione. *Internet based mobile ad hoc networking.* IEEE Internet Computing, pp: 63-70, 1999.

[7] Siva Ram, C. & Murthy, B.S. Manoj. *Ad Hoc Wireless Networks, Architectures and Protocols.* 2004.

[8] Nico Bayera, Bangnan Xua, Sven Hischkeb. *An Architecture for connecting Ad hoc Networks with the IPv6 Backbone (6Bone) using a Wireless Gateway*. European Wireless 2004, Barcelona, Spain, 2004.

[9] http://tools.ietf.org/id/draft-singh-autoconf-adp-00.txt

[10] U. Jonsson, F. Alriksson, T. Larsson, P. Johansson, and G. Maguire, Jr. *MIP MANET - Mobile IP for Mobile Ad hoc Networks.* In Proceedings of *IEEE/ACM Workshop on Mobile and Ad Hoc Networking and Computing*, Boston, MA, USA, August 1999.

[11] A. Hamidian. *A Study of Internet Connectivity for Mobile Ad Hoc Networks in NS 2.* Master's thesis. *Department of Communication Systems, Lund Institute of Technology, Lund University*. January 2003.

[12] Raghavendra, R., et al. *IPAC – An IP-based Adaptive Packet Concatenation forMultihop Wireless Networks*. in Proceedings of *Asilomar Conference on Systems, Signals and Computing*. Pacific Grove, CA. 2006.

[13] E. Nordstrøm. AODV *implementation on Linux. AODV UU.* [Online]. *Available*: http://core.it.uu.se/core/index.php/AODV-UU. 2004.

[14] C. PERKINS, E. ROYER and S. DAS. *Ad hoc On-Demand Distance Vector (AODV) Routing*, Internet Draft, draft-ietf-manet-aodv-11.txt, *work in progress*, June 2002.

[15] Björn Wiberg. *Porting AODV-UU Implementation to ns-2 and Enabling Trace- Based Simulation*. Uppsala University, December 2002.

[16] Y. Xiao and J. Rosdahl. *Throughput and delay limits of IEEE 802.11*. IEEE Commun. Lett., vol. 6, no. 8, pp. 355–357, Aug 2002.

[17] Y. Xiao and J. Rosdahl. *Performance analysis and enhancement for the current and future IEEE 802.11 MAC protocols.* ACM SIGMOBILE *Mobile Comput. Commun. Rev. (MC2R),* vol. 7, no. 2, pp. 6–19, Apr. 2003.

[18] A. Jain, M. Gruteser, M. Neufeld, and D. Grunwald. *Benefits of packet aggregation in ad-hoc wireless network.* Tech. Rep. CU-CS-960-03, *Department of Computer Science, University of Colorado at Boulder*, 2003.

[19] H. YAMADA and N. HIGUCHI. *Voice quality evaluation of IP-based voice stream multiplexing schemes.* In *Local Computer Networks (LCN)*, pp. 356.364, 2001.

[20] R. Komolafe, O. Gardner. *Aggregation of VoIP Streams in a 3G mobile network: A Teletrafic Perspective.* In *European Personal Mobile Communications Conference (EPMCC)*, 2003.

[21] J. Tourrilhes. *Packet Frame Grouping*. HP Labs *Technical Reports*, Bristol, UK, Tech. Rep. HPL-97-132, October 1997.

[22] *IEEE 802.11 : Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*. IEEE.

[23] D. Kliazovich and F. Granelli .*On Packet Concatenation with QoS support for Wireless Local Area Networks.* In Proceedings of *ICC 2005, Seoul, Korea* pp. 1395.1399, May 2005.

[24] K. Yeung. *802.11a Modeling and MAC Enhancements for High Speed Rate Adaptive Networks.* Techical Report *UCLA, Tech. Rep*. 2002.

[25] H. Zhai and Y. Fang. *A Distributed Adaptive Packet Concatenation Scheme for Sensor and Ad Hoc Networks*. In Proceedings of *Milcom, Atlantic City, NJ*, October 2005.

[26] R. Draves, J. Padhye, and B. Zill. *Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks.* In Proceedings of the *ACM MOBICOM, Philadelphia, PA, pp. 114.128.* September 2004.

[27] Kassler, A.J. and Dely, P. *On Packet Aggregation for VoIP in Wireless Meshed Networks*. in Proceedings of *7th Scandinavian Workshop on Wireless Ad-hoc Networks*. Stockholm, Sweden. 2007.

[28] D. Estrin, K. Fall, S. Floyd, J.Heidemann, A. Helmy, P. Huang, S. McCanne, K Varadhan, Y. Ya and H. Yu. *Advances in Network Simulation.* In: Computer, 33(5):59-67, May 2000.

[29] P. A. Branch and G. J. Armitage, "*Towards a General Model of First Person Shooter Game Traffic*", Technical report *Center of Advanced Internet Architectures*, Melbourne Australia, 2005.

[30] http://nsnam.isi.edu/nsnam/index.php/User_Information

[31] http://www-net.cs.umass.edu/~ratton/ns-lecture/ns-lecture.ppt

[32] Quake 3, *idsoftware*. http://www.idsoftware.com/ February2005.

[33] Gonzalo Iglesias Aguiño. *Performance of VoIP strategies for hybrid Mobile Ad Hoc Networks*. Master's thesis Karlstad University 2007.

[34] EverQuest, PlanetSide. *Verant interactive*, January 2001. http://www.verant.com/

[35] Ensemble Studios, Microsoft Game Studios 1997. http://www.microsoft.com/games/empires/

[36] International Telecommunication Union, *One-way transmission time, G.114*. International Telecommunication Union: Transmission Systems and Media, Digital Systems and Networks, 2003.

[37] Groom, F.M. and Groom, K.M., *The basics of voice over Internet protocol*. International Engineering Consortium, ISBN: 0-931695-26-0. Chicago, 2004.

[38] G.Armitage, *An Experimental Estimation of Latency Sensitivity in Multiplayer Quake3*, Proceedings of *11th IEEE International Conference on Networks (ICON)*, 2003.

[39] http://searchvoip.techtarget.com/sDefinition/0,,sid66_gci213534,00.html

[40] http://www.rezonance.co.uk/95_percentile_rule.asp

[41] http://en.wikipedia.org/wiki/Standard_deviation

[42] http://mathworld.wolfram.com/StandardDeviation.html

[43] http://en.wikipedia.org/wiki/Standard_error_(statistics)

[44] S. Zander, G. Armitage, "*Empirically Measuring the QoS Sensitivity of Interactive Online Game Players*," Proc. *Australian Telecommunications Networks & Applications Conference 2004 (ATNAC 2004)*, Sydney, Australia, December 2004.

[45] Kyungtae, K. and Sangjin, H. *VoMESH: voice over wireless mesh networks*. In Proceedings of *IEEE Wireless Communications and Networking Conference* 2006. Las Vegas, NV, USA: IEEE.

[46] Kyungtae, K., et al. *On packet aggregation mechanisms for improving VoIP quality in mesh networks*. in Proceedings of *63rd Vehicular Technology Conference* 2006. Melbourne, Vic., Australia: IEEE.

[47] Niculescu, D.; Ganguly, S.; Kim, K.; Izmailov, R. *Performance of VoIP in a 802.11 Wireless Mesh Network.* In proceedings of 25th IEEE International Conference on Computer Communications INFOCOM 2006. Barcelona April 2006.

[48] Deborah Rumsey. “Statistics For Dummies”, ISBN:0764554239, John Wiley & Sons © 2003.

6 Appendix

6.1 Common Abbreviations

MANET	Mobile Ad-Hoc Network
FPS	First Person Shooter
AODV	Ad-Hoc On Demand Distance Vector
AODV-UU	AODV – Uppsala University version
NS-2	Network Simulator version 2
PERL	Practical Extraction and Reporting Language
TCL	Tool Command Language
TCP	Transmission Control Protocol
IP	Internet Protocol
MAC	Medium Access Control
MTU	Maximum Transmission Unit
RPG	Role Playing Games
RTS	Real Time Strategy
SNR	Signal to Noise Ratio
MCS	Maximum Concatenation Size
MCI	Maximum Concatenation Interval
CBR	Constant Bit Rate
UDP	User Datagram Protocol
EXP	Exponential
CSThresh	Carrier Sense Threshold
RXThresh	Receiver Threshold
ID	Identification

Table 6.1: List of Abbreviation used in this thesis

6.2 List of Code Files

The following table contains code files from [3] that were used/modified in this thesis. A new packet type for quake3 game packets is created. Files with their little description are as follows.

Code File	Description
common/quake3pkt[cc, h]	Contains definition of QUAKE3P packets.
common/agent-deaggregator[cc, h]	Implements agent de-aggregator.
common/ipmeta[cc, h]	Contains definition of IPMETA packets.
common/packet[cc, h]	Creates IPMETA packets for QUAKE3P game packets.
tcl/lib/ns-default.tcl	Added default parameters for quake3 game server, clients and graphics card.
tcl/lib/ns-packet.tcl	Added definition for QUAKE3P packets.
queue/aggregator[cc, h]	Modified implementation of queue aggregator for game packets.

Table 6.2: List of added/modified source files in NS2

Ns2 simulation files along with game model and packet aggregation are provided on attached CD. It also contains the PERL script used for post tracing analysis.