Cross Layer Optimization and Network Congestion

Dharamvir Saini
Department of Computer Science and Engineering
Kalpana Chawla Government Polytechnic For Women, Ambala City, Haryana, India

Ashish Verma
(Asst. Prof.) Dept. of Computer Engineering
Sri Sukhmani Institute of Engg. & Technology Derabassi, Punjab, India

Abstract- A **network is nothing more than two or more computers connected by a cable or by a wireless connection so that they can communicate and exchange information or data. In other words "Network Means a collection of interconnected computer network of stand-alone computer. Commenting on the computer for the exchange of information. The connection can be over copper, fiber optic, microwave and satellite communications". As we know the major purpose of cross layer is to improve the network services or QoS (quality of services).Network congestion is the major constraint in the network services, so to improve the services we have to implement the congestion control algorithms. In this paper we have automated two congestion algorithms using net beans technology. The real benefit to automate these algorithms is we can easily calculate which algorithm is suitable in a particular situation. Thus this paper describes the simulated process of conjunction algorithms.**

Keywords— (MANET) Mobile Adhoc Networking, QoS(Quality of Services),Cross Layer and Cross layer Optimization.

I. INTRODUCTION

"The connection can be over copper, fiber optic, microwave and satellite communications". Obviously, computers can exchange information in other ways called the sneaker net or Floppy Net means that to copy a file to a diskette and then walk the disk over to some other computer. We can create a computer network by hooking all the computers in the office together with cables and installing a special network interface card (NIC) in each computer so we have a place to plug in the cable. Then we set up the computer's operating-system software to make the network work. If we don't want to mess with cables, we can create a wireless network instead. In a wireless network, each computer is equipped with a special wireless network adapter.

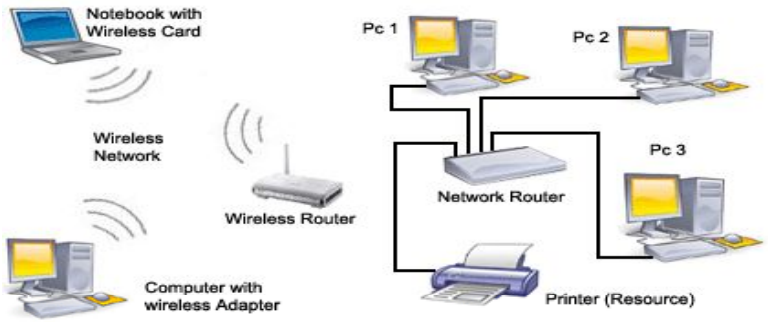

Figure 1. Network Structure(Wired and Wireless).

Above figure shows a simple network with three computers and a printer. We can see that all devices are connected with network cables to a central network device called a Network Router. The printer in this network can be used by all the PCs. Also the figure show we how the Wireless network Works, the Notebook and the Computer connected with the wireless router by wireless adapters which equipped with them.

II. MOBILE AD-HOC NETWORK

The past decade has shown a phenomenal growth in wireless communications. Cellular systems have been standardized and Personal Communication Services (PCS) and the 3rd generation radio technology are being used providing wide-band services to mobile users.

Additionally, wireless networking is being used more and more in both fixed and mobile usage scenarios, whereas high quality multimedia (voice, video and data) services over high-speed wireless local area networks (LANs) are becoming a reality. Wireless LANs (e.g. HiperLAN2, IEEE 802.11), being interconnected to a fixed network, are offering up to 54Mbps both to residential and business environments with high quality of service (QoS). The demand of these multimedia applications has been largely witnessed so far in fixed networks but as life style is rapidly changing, internet-like applications are more and more attractive to mobile users as well. In parallel with (and separately from) the single hop model for today's cellular/wireless communications, another type of model based on radio to radio multihopping, has been evolving to serve a growing number of applications which rely on a fast deployable, multihop, wireless infrastructure. A multihop mobile radio network, also called mobile ad hoc network (MANET) is a self-organizing and rapidly deployable network in which neither a wired backbone nor a centralized control exists. The network nodes communicate with one another over scarce wireless channels in a multi-hop fashion. The ad hoc network is adaptable to the highly dynamic topology resulted from the mobility of network nodes and the changing propagation conditions. MANETs are a new paradigm of wireless wearable devices enabling instantaneous person-to-person, person-to-machine or machine-to-person communications immediately and easily. Possible commercial applications include business associates sharing information during a meeting, students using laptop computers to participate in an interactive lecture, and emergency disaster relief personnel coordinating efforts in natural disasters. In these applications, where a fixed backbone is not available, a readily deployable wireless network is needed. Mobile ad hoc networks are also a good alternative in rural areas or third world countries where basic communication infrastructure is not well established. Another interesting application of mobile ad hoc networks is ubiquitous computing. Intelligent devices are connected with one another via wireless links and are self-organized in such a way that a newly joined node can request service from local servers without any human intervention. When designing mobile ad hoc networks, several interesting and difficult problems arise due to shared nature of the wireless medium, limited transmission power (range) of wireless devices, node mobility, and battery limitations. The limited transmission range of wireless network interfaces coupled with the highly dynamic routing infrastructure due to mobility create a lot of concerns when addressing issues such as dynamic routing, efficient channel access and quality-of-service (QoS) support.

III. CROSS LAYER

The transition from wired to wireless networks opened up new horizons for research. There is a multitude of emerging network applications designed for personal and mobile devices. To support such a variety of applications, there is an ever growing trend to optimize the performance of the wireless networks. However, the communication stack which handles the end to end communication in data networks was initially designed for wired networks. During the transition from wired to wireless only a few protocols at the underlying layers in the communication stack are replaced with new protocols for wireless networks. In other words, the higher layers and their residing protocols remained ignorant of the fact that now they are operating without a wire. This lack of knowledge about the new protocols and different characteristics of a new physical layer (PHY) consequently caused wrong assumptions at the higher layers. While layer to layer abstraction was a goal of the layered protocol suite; in case of wireless networks it presented a few problems. In order to cope with these problems an idea of cross layer information exchange was coined in the research community. The idea behind cross layer information exchange is to use various parameters from different layers for joint optimization of protocols across the communication stack. Some of the research issues dealt with the help of Cross layer communication are:

- Application layer adaptation based on cross layer strategies.
- Cross layer design framework for real-time multimedia streaming
- Cross layer content delivery architecture
- Complexity and scalability issues in cross layer design
- Signaling for cross layer protocol interaction
- Scheduling Algorithms and Link Adaptation
- Interactions among PHY/MAC/RLC and transport layer protocol
- Cross layer adaptation for energy minimization in wireless networks
- MAC protocols with multimedia QoS support in wireless networks

- Transport/streaming protocols for end-to-end QoS support
- Applications of network coding in cross layer design
- Protocols' implementation, analysis of correctness and efficiency, and interoperability with legacy system
- Energy saving and power control protocols for ad hoc and sensor networks
- Cross-layer strategies in 2G/3G/4G cellular system, wireless sensor, ad hoc networks and other emerging communications.
- Cross-layer design and implementation in Software defined and cognitive radio.

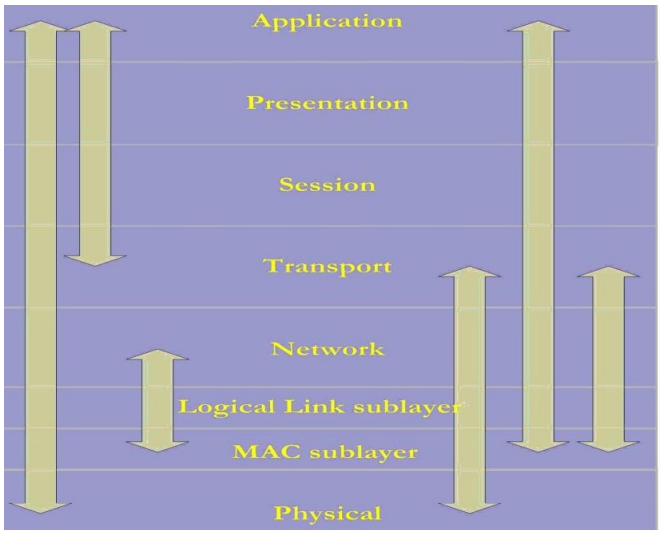

Figure 2. Cross Layer Design

IV. CROSS-LAYER OPTIMIZATION

Cross-layer optimization is an escape from the pure waterfall-like concept of the OSI communications model with virtually strict boundaries between layers. The cross layer approach transports feedback dynamically via the layer boundaries to enable the compensation for e.g. overload, latency or other mismatch of requirements and resources by any control input to another layer but that layer directly affected by the detected deficiency. In the original OSI networking model, strict boundaries between layers are enforced, where data are kept strictly within a given layer. Cross layer optimization removes such strict boundaries to allow communication between layers by permitting one layer to access the data of another layer to exchange information and enable interaction. Especially in information routing with concurrent demand for limited capacity of channels there may be a need for a concept of intervention to balance between e.g. the needs of intelligible speech transmission and of sufficiently dynamic control commands. Any fixed allocation of resources will lead to a mismatch under special conditions of operations. Any highly dynamic change of resource allocation might affect the intelligibility of voice or the steadiness of videos. However, as with other optimizing strategies, the algorithm consumes time as well.

V. PROPOSED ALGORITHM

Type of Congestion

There are two types of congestion: *transient* and *persistent*. Transient congestion can be managed with a queue of buffer at the router. During to the congestion period the queue will grow and contain the excess packets. When the congestion period ends, the buffered data is forwarded to the appropriate output link. On the other hand, persistent congestion is said to occur when the data overflows the buffer. While transient congestion only introduces a delay in data transmission, persistent congestion results in data loss. These problems are tackled in two ways. Either the router detects the queue build up and informs the sources to decrease their transmission rate. Such a strategy is called *Congestion avoidance.* The other method is to use end-to-end strategies where the routers do not get directly involved but the hosts use indirect methods to detect congestion. Such a mechanism is known as *Congestion*

Control. In TCP/IP, the most commonly used protocol in the Internet, both methods are used to tackle problem related to congestion. Iit is pointed out that the very detection of occurrence of congestion in a network may not be very easy. The immediate side effects of congestion are data loss or unacceptable delay in transmission. However, these may be results of faulty routers or corrupt packets. Hence these do not necessarily form good indices for congestion detection. Also, delay or loss of performance is very much a qualitative measure. Delay may be acceptable for one type of user and not to another. However, proper congestion control mechanisms do result in better network performance under heavy load conditions.

THE TOKEN BUCKET(LEAKY BUCKET) MODEL

This chapter tries to explain basic token bucket model, its parameters and its usage in diffuser environment. Token bucket represents the ***Policing*** function of Traffic Conditioning Block of diffuser. A token bucket flow is defined by (r,b), r denotes the rate at which tokens(credits) are accumulated and b is the depth of the token pool(in bytes).

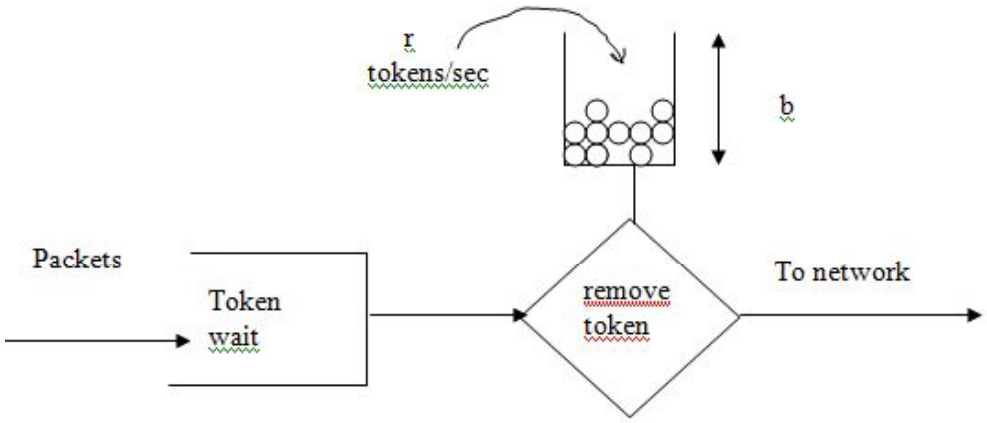

Figure 3. Leaky Bucket

As it shown above the new token are adding to the bucket at rate of r tokens/sec, the maximum token can be accumulated is b bytes. If the bucket is full, the incoming tokens will be thrown away. The Token Bucket(TB) profile contains three parameters: an average rate, a peak rate, and burst size. Lets look at the meaning of these parameters:

Average rate: The network may wish to limit the long-term average rate at which a flow's packets can be sent into the network. The important point here is the interval of time over which the average rate will be policed. For example; a flow which has 10000 packets/sec is more flexible than a flow which has 10 packets/msec. Even though both have the same average rate, their behaviors are different. The flow which has 10000 packets/sec can send 100 packets in a one msec long-term, but the flow which has 10 packets/msec cannot sent 100 packets in a one msec long-term.

Peak rate: Defines the maximum rate at which packets can be sent in short interval time. In above example; even though the flow's long term average rate is 10000 packets/sec peak rate can be limited to 200 packets/msec.

Burst size: Burst size is the maximum number of packets which can be transmitted extremely in a short interval time. In other word, when time interval approaches zero, the burst size limits the number of packets which can be

transmitted to the network. when we don't consider time interval as zero, burst size depends on bucket depth b, and maximum speed of output link . We will clarify in the next section in more detail.

In any interval t [S,S+t], let say in time S, and the number of tokens in the bucket are a(a<=b), and during time interval [S,S+t] the counter accumulate rt tokens more. The total number of tokens available for source to transmit packets is a+rt, (a+rt<=b+rt). When the bucket size is b and during interval time [S,S+t], b+rt tokens accumulated. In this case bucket gets empty by the maximum rate of output link. If we call maximum speed of output *B.* The time which buckets gets empty b+rt=*B*t ➔ t=b/(*B*-r). Choosing all these parameters can be a bit tricky. One of the potential problem with token bucket is it allows burst again. If another burst come during this time interval, it cannot be handled, or in the case of bursts come very often the output speed will be maximum for the period of burst which can cause congestion. This problem can be reduced by carefully choosing parameters.
Essentially, the network has to simulate the algorithm and make sure that no more packets or bytes are being sent than are permitted. Depends on the agreements the excess packets can be dropped, remarked etc.

When we concern about delay and reliability, we could think some other aspects of the parameters. If we consider single flow, delay can be expressed as

Delay=b/R+C/R +D R is the maximum transmit rate of output link, D is non-rate-dependent delay, C is the delay which depends on flow transmission rate. C and D are local parameters. The delay in node must be less than the above value. The end-to-end delay for this flow is the sum of all the delay through the route.

A. Token Bucket in Differentiated Services

The operation of the Token Bucket in diffserv can be defined for as follow:

- Arriving packets of L bytes are conforming (immediately processed) if there are at least L tokens in the bucket(one token= one byte)
- If the current number of accumulated tokens b' is less than the arriving number of packets L, L-b' packets are nonconforming.
- Packets are allowed to the average rate in bursts up to the burst size, as long as they do not exceed the peak rate, at which point the bucket is drained.
- If there are no packets to be transmitted, tokens can be accumulated up to size of b. The rest of tokens will be thrown away.

Here conforming and non-conforming functions are depends on SLA agreement.
If the packets are non-conforming the following actions can be taken.

- The packet may be thrown away.
- The packet may be re-marked in a particular way.
- The packet can be buffered(by inserting a buffer between the inflow and the decision point) and not released until sufficient number of tokens arrive in the bucket.

1) Token Bucket in the QPS

The QPS has the following features.

- No loss due to congestion.
- No latency guaranties
- Worst case jitter bounds

Because of the above conditions, the packets which are no-conform will be dropped. Choosing parameters (r,b) in QPS a little bit different than other services such as assured services.

In QPS The Token Bucket depth(b) should be limited to the equivalent of only one or two packets. When a QPS packet arrived, if there is a token available, it will be forwarded immediately. But when there is no token available, the QPS packets must wait until a token arrives. If QPS flow bursts enough to overflow the holding queue, its packets will be dropped. During the presentation of the flow to the network, the size of holding queue can be

defined. Un-configured holding queues should be capable of holding at least two bandwidth-delay products, adequate for TCP connections.

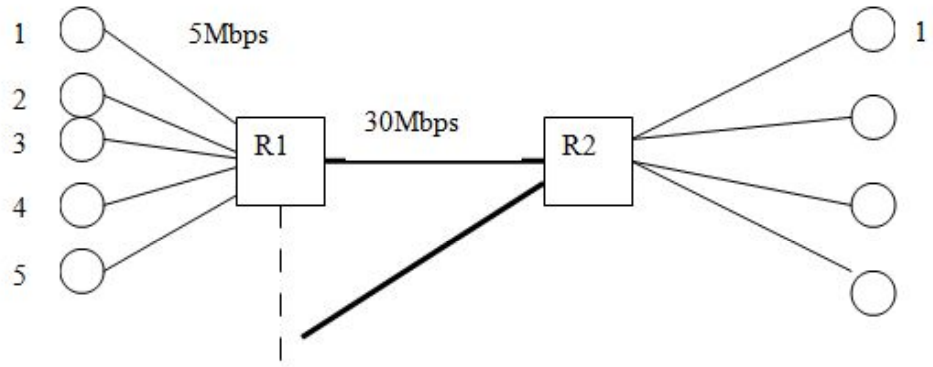

Figure 4. Token Bucket in the QPS

In Figure 4, Suppose all the hosts which connected to router R1 are sending packets at average rate of 5Mbps, if the token bucket depth is higher than what we defined above, that will allow bursts of packets passing through a router. But letting one burst passing through one router is not a problem, problems may occur when multiple bursts arrive simultaneously at a router and compete for the same output link, when that happens, there can cause long queuing delays or even packet drops due to buffer overflow. Also even queuing is allowed, we cannot guaranties end-to-end QoS for QPS in this case.

Therefore token bucket for QPS should be similar to Figure 5.

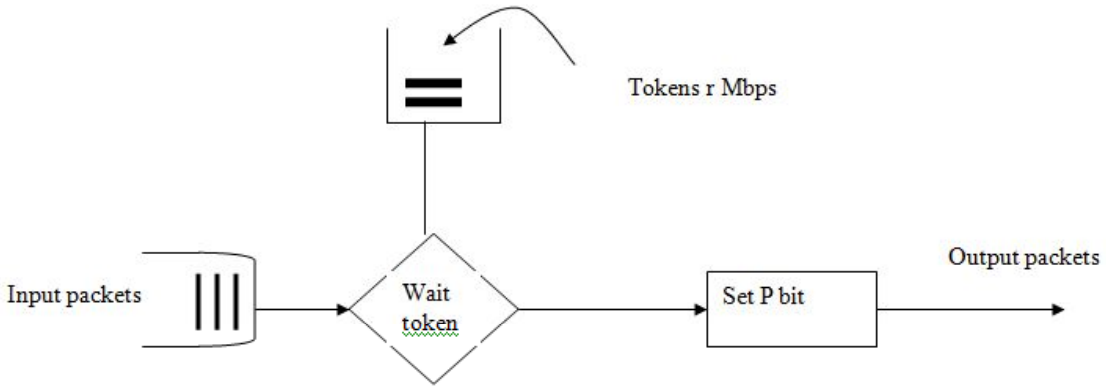

Figure 5. Token bucket

As it defined in the SIBBS, QPS must have higher priority than other classes. Each output interface of router must have at least two queues. One queue for QPS one for best effort. In the border router traffic should be shaped according to the SLA. For shaping and queuing Round Robin Queuing and WFQ does not satisfy our requirement. Class based Queuing(CBQ) fits to our requirement better than others.
CBQ is an IP feature that classifies traffic according to very granular network policies. It allows individual applications, subnets, or different groups of users to each receive bandwidth tailored to meet their specific QoS

requirements. Bandwidth guarantees and borrowing privileges can be applied to each traffic class in real time, dynamically, by using CBQ.

Floyd-Warshall Algorithm

A weighted graph is a collection of points(vertices) connected by lines(edges), where each edge has a weight(some real number) associated with it. One of the most common examples of a graph in the real world is a road map. Each location is a vertex and each road connecting locations is an edge. We can think of the distance travelled on a road from one location to another as the weight of that edge.
Given a weighted graph, it is often of interest to know the shortest path from one vertex in the graph to another. The Floyd-Warshall algorithm algorithm determines the shortest path between all pairs of vertices in a graph. The the vertices in a graph be numbered from 1 to n. Consider the subset {1,2...,k} of these n vertices. Imagine finding the shortest path from vertex i to vertex j that uses vertices in the set {1,2, ...,k} only.
There are two situations:
1) k is an intermediate vertex on the shortest path.
2) k is not an intermediate vertex on the shortest path.

In the first situation, we can break down our shortest path into two paths: i to k and then k to j. Note that all the intermediate vertices from i to k are from the set {1,2,...,k-1} and that all the intermediate vertices from k to j are from the set {1,2,...,k-1} also.

In the second situation, we simply have that all intermediate vertices are from the set {1,2,...,k-1}.

Now, define the function D for a weighted graph with the vetrtices {1,2,...n} as follows:

D(i,j,k) = the shortest distance from vertex i to vertex j using the intermediate vertices in the set {1,2,...,k}

Now, using the ideas from above, we can actually recursively define the function D:

$$D(i,j,k) = w(i,j), \text{ if } k=0$$
$$\min(D(i,j,k-1), D(i,k,k-1)+D(k,j,k-1)) \text{ if } k > 0$$

we do not allow intermediate vertices, then the shortest path between two vertices is the weight of the edge that connects them. If no such weight exists, we usually define this shortest path to be of length infinity.
The second line pertains to allowing intermediate vertices. It says that the minimum path from i to j through vertices {1,2,...,k} is either the minimum path from i to j through vertices {1,2,...,k-1} OR the sum of the minimum path from vertex i to k through {1,2,...,k-1} plus the minimum path from vertex k to j through {1,2,...k-1}. Since this is the case, we compute both and choose the smaller of these.

All of this points to storing a 2-dimensional table of shortest distances and using dynamic programming for a solution.

Here is the basic idea:

1) Set up a 2D array that stores all the weights between one vertex and another. Here is an example:

0	3	8	inf	-4
inf	0	inf	1	7
inf	4	0	inf	inf
2	inf	-5	0	inf
inf	inf	inf	6	0

Notice that the diagonal is all zeros.
Now, for each entry in this array, we will "add in" intermediate vertices one by one, (first with k=1, then k=2, etc.) and update each entry once for each value of k.

After adding vertex 1, here is what our matrix will look like:

0	3	8	inf	-4
inf	0	inf	1	7
inf	4	0	inf	inf
2	5	-5	0	-2
inf	inf	inf	6	0

After adding vertex 2, we get:

0	3	8	4	-4
inf	0	inf	1	7
inf	4	0	5	11
2	5	-5	0	-2
inf	inf	inf	6	0

After adding vertex 3, we get:

0	3	8	4	-4
inf	0	inf	1	7
inf	4	0	5	11
2	-1	-5	0	-2
inf	inf	inf	6	0

After adding vertex 4, we get:

0	3	-1	4	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

Finally, after adding in the last vertex:

0	1	-3	2	-4
3	0	-4	1	-1
7	4	0	5	3
2	-1	-5	0	-2
8	5	1	6	0

Looking at this example, we can come up with the following algorithm:

Let D1 store the matrix with the initial graph edge information. D2 will stored calculated information look at D1.
Path Reconstruction in Floyd's Algorithm

When you run Floyd’s, what you’re really doing is initializing your distance matrix and path matrix to indicate the use of no immediate vertices. Thus, you are only allowed to traverse direct paths between vertices.

At each step of Floyd’s, you essentially find out whether or not using vertex k will improve an estimate between the distances between vertex i and vertex j. If it does improve the estimate here’s what you need to record:

1) record the new shortest path weight between i and j
2) record the fact that the shortest path between i and j goes through k

#1 is done in the edited adjacency matrix. Hopefully this update with the if statement is fairly easy to understand.

Here is how #2 is done:

First, it's important to understand what the path matrix stores. In particular, when

path[i][j] = k, that means that in the shortest path from vertex i to vertex j, the LAST vertex on that path before you get to vertex j is k.

Based on this definition, we must initialize the path matrix as follows:

path[i][j] = i if i!=j and there exists an edge from i to j
= NIL otherwise

The reasoning is as follows:

If you want to reconstruct the path at this point of the algorithm when you aren't allowed to visit intermediate vertices, the previous vertex visited MUST be the source vertex i. NIL is used to indicate the absence of a path.

Dijkstra's Algorithm

This algorithm finds the shortest path from a source vertex to all other vertices in a weighted directed graph without negative edge weights. Here is the algorithm for a graph G with vertices V = $\{v_1, \ldots v_n\}$ and edge weights w_{ij} for an edge connecting vertex v_i with vertex v_j. Let the source be v_1. Initialize a set S =∅. This set will keep track of all vertices that we have already computed the shortest distance to from the source. Initialize an array D of estimates of shortest distances. D [1] = 0, while D[i] =∞, for all other i. (This says that our estimate from v_1 to v_1 is 0, and all of our other estimates from v_1 are infinity.) Imagine that you want to figure out the shortest route from the source to all other vertices. Since there are no negative edge weights, we know that the shortest edge from the source to another vertex must be a shortest path. (Any other path to the same vertex must go through another, but that edge would be more costly than the original edge based on how it was chosen.) Now, for each iteration, we try to see if going through that new vertex can improve our distance estimates. We know that all shortest paths contain sub paths that are also shortest paths. (Try to convince yourself of this.) Thus, if a path is to be a shortest path, it must build off another shortest path. That's essentially what we are doing through each iteration, is building another shortest path. When we add in a vertex, we know the cost of the path from the source to that vertex. Adding that to an edge from that vertex to another, we get a new estimate for the weight of a path from the source to the new vertex.
This algorithm is greedy because we assume we have a shortest distance to a vertex before we ever examine all the edges that even lead into that vertex. In general, this works because we assume no negative edge weights. The formal proof is a bit drawn out, but the intuition behind it is as follows: If the shortest edge from the source to any vertex is weight w, then any other path to that vertex must go somewhere else, incurring a cost greater than w. But, from that point, there's no way to get a path from that point with a smaller cost, because any edges added to the path must be non-negative. By the end, we will have determined all the shortest paths, since we have added a new vertex into our set for each iteration.This algorithm is easiest to follow in a tabular format.

The adjacency matrix of an example graph is included below. Let a be the source vertex.

	a	b	c	d	e
a	0	10	inf	inf	3
b	inf	0	8	2	inf
c	2	3	0	4	inf
d	5	inf	4	0	inf

e	inf	12	16	13	0

Here is the algorithm:

Estimates				
	b	c	d	e
Add to Set				
a	inf	inf	3	10
e	19	16	3	10
b	18	12	3	10
d	16	12	3	10

We changed the estimates to c and d to 19 and 16 respectively since these were improvements on prior estimates, using the edges from e to c and e to d. But, we did NOT change the 10 because 3+12, (the edge length from e to b) gives us a path length of 15, which is more than the current estimate of 10. Using edges bc and bd, we improve the estimates to both c and d again. Finally using edge dc we improve the estimate to c.
Now, we will prove why the algorithm works. We will use proof by contradiction. After each iteration of the algorithm, we "declare" that we have found one shortest path. We will assume that one of these that we have found is NOT a shortest path.

Let t be the first vertex that gets incorrectly placed in the set S. This means that there is a shorter path to t than the estimate produced when t is added into S. Since we have considered all edges from the set S into vertex t, it follows that if a shorter path exists, its last edge must emanate from a vertex outside of S to t. But, all the estimates to the edges outside of S are greater than the estimate to t. None of these will be improved by any edge emanating from a vertex in S (except t), since these have already been tried. Thus, it's impossible for ANY of these estimates to ever become better than the estimate to t, since there are no negative edge weights. With that in mind, since each edge leading to t is non-negative, going through any vertex not in S to t would not decrease the estimate of its distance. Thus, we have contradicted the fact that a shorter path to t could be found. Thus, when the algorithm terminates with all vertices in the set S, all estimates are correct.

Trying an example on the graph with the following adjacency matrix using a as the source.

	a	b	c	d	e
a	0	3	4	inf	inf
b	inf	0	inf	6	8
c	inf	2	0	1	5
d	inf	inf	inf	0	2
e	inf	inf	inf	inf	0

Path Reconstruction in Dijkstra's Algorithm

Given the history of the values in the distance estimate array, we can trace the shortest path. In the example below, consider determining the shortest distance from a to c:

Estimates

Add to Set	b	c	d	e
a	10	inf	inf	3
e	10	19	16	3
b	10	18	12	3
d	10	16	12	3

When we updated the estimate to c to be 16, we added vertex d. This means that the last edge of the path was d -> c. Now, go to the column for d. We see that d's distance was updated when we added b to the set S. Thus, the last edge of the shortest path from a to d is the edge d->b. Finally, we go to the column for b and find that the shortest distance to b is obtained by the edge a->b. Thus, putting everything together, we have the path a->b->d->c.

VI. EXPERIMENT AND RESULT

IMPLEMENTATION OF LEAKY BUCKET ALGORITHM

Traffic management is an important function in high speed networks. The leaky bucket algorithm is a general algorithm that can be electively used to police real time traffic. Both Frame Relay and Asynchronous Transfer Mode (ATM) networks use a form of the leaky bucket algorithm for traffic management. ATM networks use a continuous state version of the leaky bucket algorithm called the Generic Cell Rate Algorithm (GCRA) to police traffic at the entrance to the ATM network.

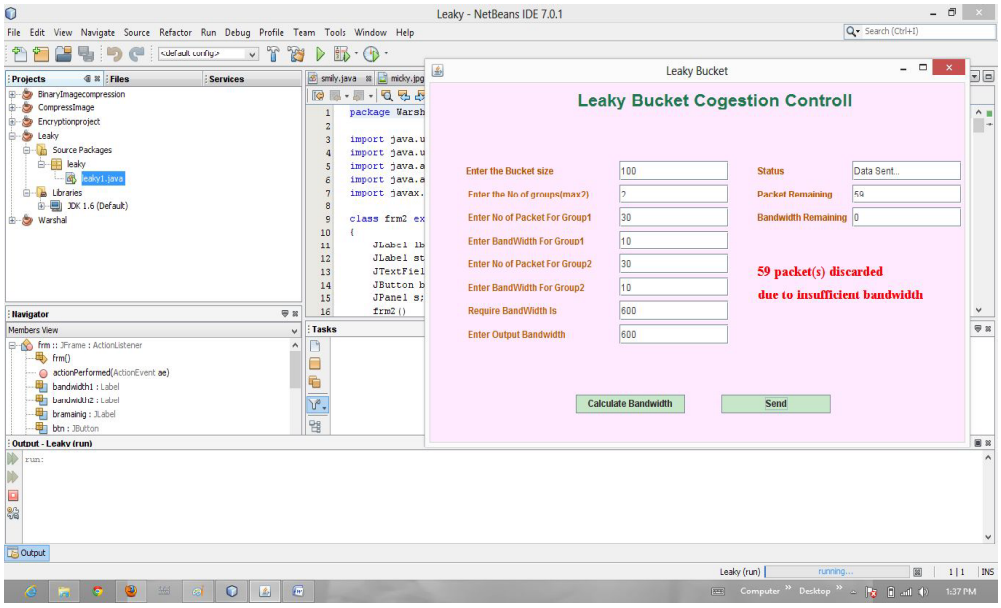

Figure 6. Implementation of Leaky Bucket Algorithm

For example in our program there are two groups who can share the bucket.

Steps to run the program

1. First of all we specify the bucket size in the first text box. For example: 10

2. Then we specify the number of groups in the second of group maximum 2.we can specify 1 and 2.
3 In the third text box we specifies the number of packets for group1 for exp: 4 and bandwidth in the fourth text box for example 10.
4. Same as the second group.
5. Click on the calculate button…and it will show the require bandwidth.
6. And then enter the output bandwidth in the text box and click on the send button.
7. It will show the desire output.

IMPLEMENTATION OF DIJKSTRA ALGORITHM

Figure 7. Implementation of Dijkstra Algorithm

This Algorithm is used to find the shortest path from one node to another and tell us the minimum cost. Step to execute the program.

1. This program is used to find the shortest path from a graph with four nodes(i.e. graph contains 4 Node)
2. Enter the cost for nodes in the text box.For example:- we can enter cost1-->1 =10 and cost2-->2=20 and so on.
3. Now enter the source node like 1 or 2.
4. Enter the ending node like 3.

Click at the Find path button and it will show the shortest path.

WARSHAL,s ALGORITHM

The warshal's algorithm is used to find the shortest path from the one node to another. In our program we will specify the starting node and ending node and program will tell us the shortest path from the starting to end node.

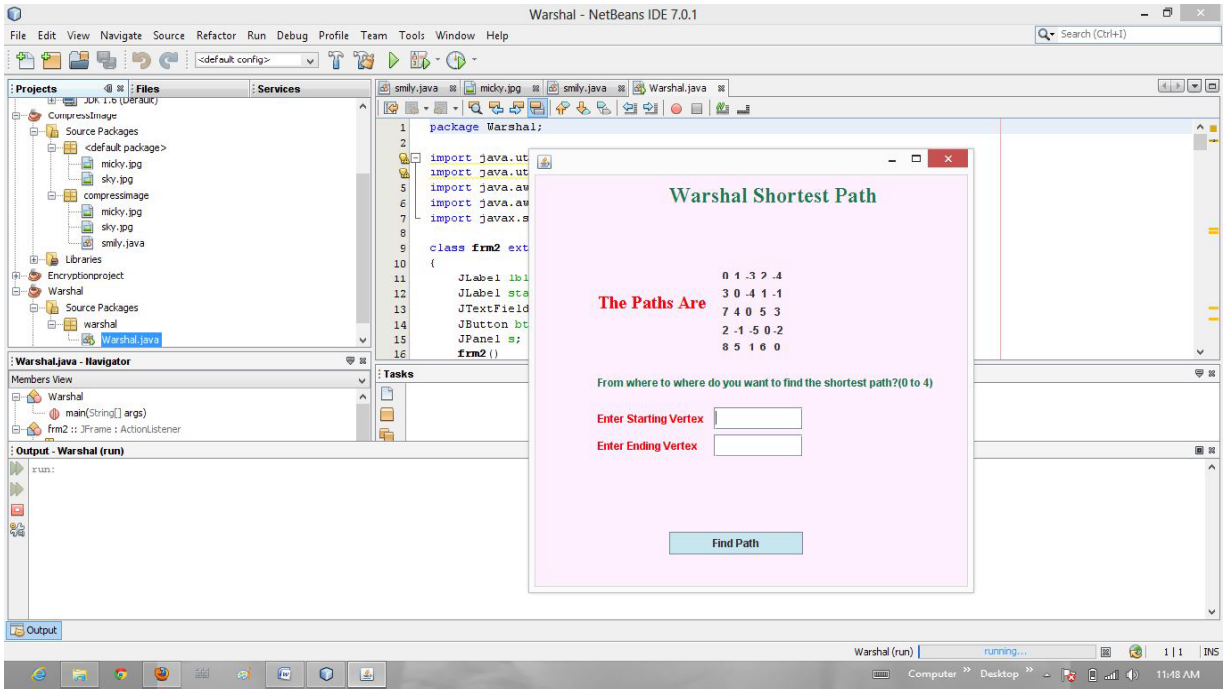

Figure 8. Implementation of Warshal's Algorithm

Step to run the program.
1. We have given a matrix of paths.
2. We specify the starting node in the first text box and ending node in the second text box.
3. And click at the FIND PATH button it will show us the shortest path.
(Note: The starting and ending node must be between 0 and 4)

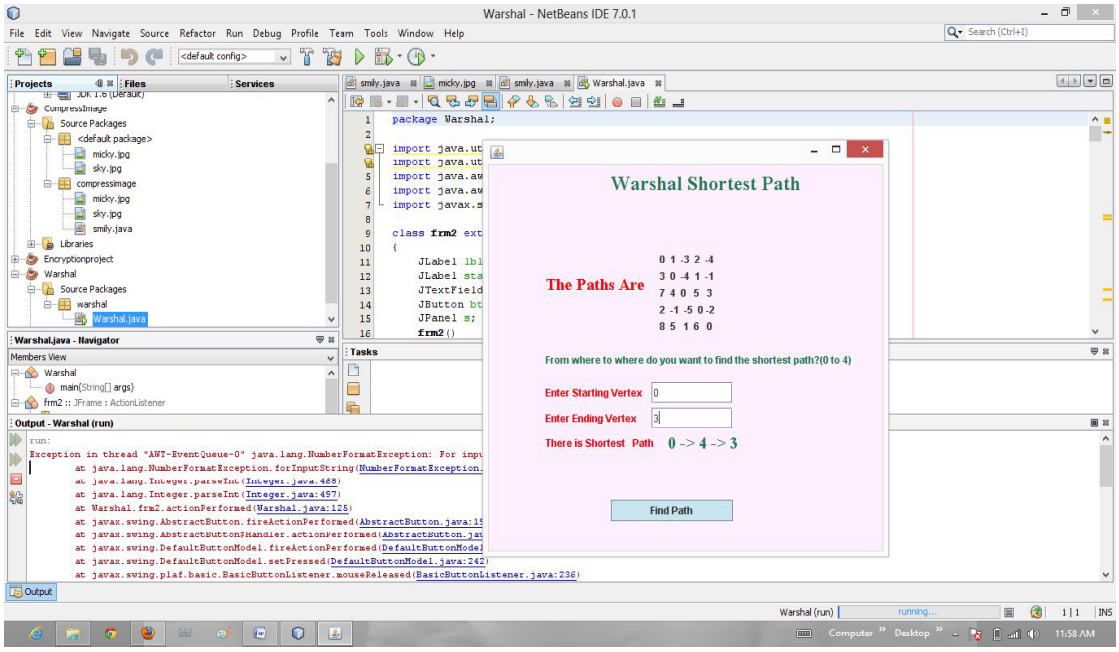

Figure 9. Implementation of Warshal's Algorithm

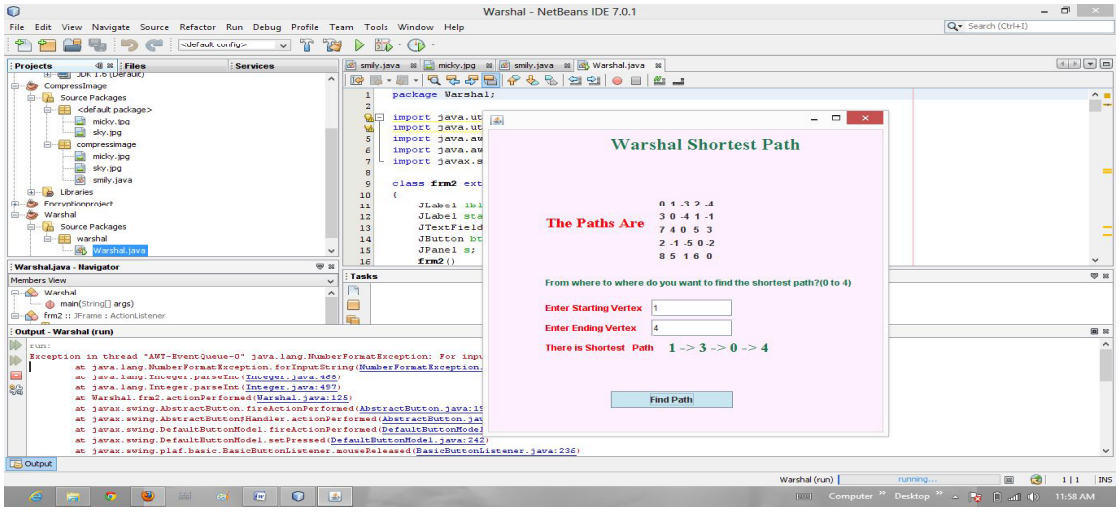

Figure 10. Implementation of Warshal's Algorithm

IV.CONCLUSION

Cross-layer optimization shall contribute to an improvement of quality of services under various operational conditions. Such adaptive quality of service management is currently subject of various patent applications; the cross-layer control mechanism provides a feedback on concurrent quality information for the adaptive setting of control parameters. Thus the current achievement of this study is to simulate & automate the congestion control algorithms .To automate the algorithms we have use the Net beans Technology and Implementation of these algorithms is user interactions based a user can the give the input to the algorithms and can see the output in the same interface after processing thus by doing this one has no need to calculate the algorithms manually. Secondly by

using this automated tool one can easily decide which algorithms is suitable in a particular situation, thus comparative study of congestion control algorithms is also possible through this implemented work. Thus at last we can conclude that successful networks are those which provide the data and information as required or take less time to complete the request these two parameters decides the status of quality of services of a network. Thus such kind of implementations really improves the quality of services of a computer network.The future of this study is very valuable and bright we can develop more automated by using new technologies like Net beans and by doing this we can improve the quality of network services, reduce the response time, calculate the node to node minimum distance easily and compare of existing algorithms and techniques can be possible easily by such kind of automated tools.

REFERENCES

[1] Ajay Todimala, "Cross-layer Reconfiguration for Surviving Multiple-link Failures in Backbone Networks," Optical Society of America OCIS codes: (060.4250) Fiber optics and optical communicationns, Networks, 2009.

[2] Mehmet C. Vuran;" Cross-layer Packet Size Optimization for Wireless Terrestrial, Underwater, and Underground Sensor Networks"; This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2008 proceedings.

[3] Mohammad Abdur Razzaque;" Cross-Layer Architectures for Autonomic Communications"; Journal of Network and Systems Management (_c 2006) DOI: 10.1007/s10922-006-9051-8

[4] Mehmet C. Vuran;" Cross-layer Packet Size Optimization for Wireless Terrestrial, Underwater, and Underground Sensor Networks "; This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE INFOCOM 2008 proceedings.

[5] Yalin Evren Sagduyu;" Cross-Layer Optimization of MAC and Network Coding in Wireless Queueing Tandem Networks"; IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 54, NO. 2, FEBRUARY 2008

[6] Keheng Huang; "Cross-layer Optimized Placement and Routing for FPGA Soft Error Mitigation " National Natural Science Foundation of China (NSFC) under grant No. (61076018, 60803031, 60633060, 60921002,

[7] Vijay T. Raisinghani;" Cross-layer design optimizations in wireless protocol stacks"; V.T. Raisinghani, S. Iyer / Computer Communications 27 (2004) 720–724; www.elsevier.com/locate/comcom

[8] Lei Shu;" Cross Layer Optimization for Data Gathering in Wireless Multimedia Sensor Networks within Expected Network Lifetime" Journal of Universal Computer Science, vol. 16, no. 10 (2010), 1343-1367 submitted: 8/7/08, accepted: 9/3/10, appeared: 28/5/10 © J.UCS

[9] E. Endroyono;" cross-layer optimization performance of single cell millimeter wave ofdm wireless network under rain fading"; Seminar Nasional Aplikasi Teknologi Informasi 2009 (SNATI 2009) ISSN:1907-5022 Yogyakarta, 20 Juni 2009

[10] Alice Combernoux;" Cross-layer Optimization of a Multimedia Streaming System via Dynamic programming" Author manuscript, published in "IEEE Int. Conf. on Image Processing, United States (2012)"; hal-00721543, version 1 - 27 Jul 2012