Node Feedback TCP Based Mechanism for Mobile Ad-hoc Network

Farooq Alam

A thesis submitted in partial fulfilment of the requirements of Liverpool John Moores University for the degree of Master of Philosophy

July 2014

List of Publications

This research is resulted in the following publications.

Node Feedback Based TCP Scheme for Mobile Ad-hoc Network, the 11th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet 2010) 21-22 June 2010, Liverpool, UK

Node Feedback Based TCP Scheme for Mobile Ad-hoc Network, GESJ: Computer Science and Telecommunications 2011|No.2 (31)

Acknowledgement

I am grateful to Allah SWT who has showered his extreme kindness by enabling me to complete this project. I am thankful to my supervisors Dr. Robert Askwith and Professor Madjid Merabti for their continuous support throughout this project.

My mother prayers and my wife and my two daughters' moral support were of great assistance in my work. I would also like to pay special thanks to my sisters, friends and colleagues for their encouragements.

Special thanks to technical and administrative staff for their support and help.

Abstract

A mobile ad-hoc network is an autonomous system of mobile nodes establishing a network in the absence of any fixed infrastructure. Mobile ad-hoc network due to potentially high mobility have provided new challenges by introducing special consideration differentiating from the unique characteristics of the wireless medium and the dynamic nature of the network topology. Due to unique network formation, routing in mobile ad-hoc network is a challenging issue. Effort has been undergoing to transform TCP so that it could support routing function in an ad-hoc network. This research has discovered that most of the TCP based variant routing solutions of mobile ad-hoc network has not been successful in addressing problem at full. Taking TCP based routing solution as a main problem, this research has proposed a novel routing solution called Node feedback TCP based mechanism as a routing scheme for mobile ad-hoc network.

Node feedback TCP based mechanism introduces a new flavor of TCP for mobile ad-hoc network. It follows an intermediate approach in between some of the existing mechanisms of TCP based schemes for mobile ad-hoc network. We have addressed TCP slow start mechanism in the context of mobile ad-hoc network and introduce measures through whom TCP can differentiate between real congestion and congestion assumed by TCP due to packet lost or route failure in mobile ad-hoc network. In addition our proposed mechanisms also deal with out-of-order delivery problem of TCP in mobile ad-hoc network. It is important to mention that NFBTCP not only address TCP related issues but also provides a number of different operations to assists in the smooth running of an ad-hoc network.

The scheme has been developed in Java and Evaluated in SWANS. In the light of the simulation experiments, it could be seen that NFBTCP performed well in all simulation environment. It can be confirmed that NFBTCP has proven itself as a fully functional and operationalable for mobile ad-hoc network, thus should be seen or taken as a new novel TCP based solution for mobile ad-hoc network. A higher number of route requests and route replies representing networking activities were observed with the increase of mobile nodes. In addition to the messages activities, good numbers of routes were added at the end of each simulation cycle. It is quite understandable that the more routes available for data transfer in mobile ad-hoc network, the better. Moreover, such additions

to the available routes could directly impact overall throughout. Lastly, nodes in mobile ad-hoc network suffer with limited resources. That makes conservation of all such resources an important issue in the context of mobile ad-hoc network. The results of simulation experiments validate the main concepts of the scheme especially congestion avoidance and out-of-order packet delivery. The scheme generates a higher number of routes suggesting that the implemented congestion avoidance and out-of-order packet delivery mechanisms of NFBTCP are successful in reducing the impact of link breakage since subsequently it was not possible for more routes to be added. The addition of more routes demonstrates that more packets are broadcast and suggests smooth flow of data and control packets. We believe NFBTCP offers a complete and an effective TCP based routing solution for mobile ad-hoc network.

List of Abbreviation

NFBTCP	Node FeedBack TCP based mechanism
TCP	Transmission Control Protocol
MANET	Mobile Ad-hoc Network
SWANS	Scalable Wireless Ad-hoc Network Simulator
JiST	Java in Simulation Time
AODV	Ad-hoc On-demand Distance Vector Routing
BER	Bit Error Rate
CWND	Congestion Window
ACK	Acknowledgement
RTO	Retransmission Time Out
OSI	Open System Interconnection
RFN	Route Failure Notification
RRN	Route Re-establishment Notification
ATCP	Ad-hoc Transmission Control Protocol
TCP-F	TCP Feedback
TCP-ELFN	TCP Explicit Link Failure Notification
ICMP	Internet Control Message Protocol
AFP	Ad-hoc Formation Packet
AACK	Ad-hoc Acknowledge Packet
JAP	Joining Ad-hoc Packet
AP	Acknowledgement Packet
LFD	Link Failure Detection
LFUP	Link Failure Update Packet
LUP	Link Update Packet
LCD	Link Capacity Detection
LFP	Link Formation Packet
BUP	Buffer Update Packet

Chapter 1. Introduction

1.1. Introduction

A wireless network allows a more flexible model of communication than a traditional network since the user is not limited to a fixed physical location. Wireless networks can further be categorized into one of two types, wireless fixed network and mobile ad-hoc network. A mobile ad-hoc network has become increasingly important because of their promise of ubiquitous connectivity beyond traditional fixed network. A mobile ad-hoc network has provided new challenges including limited power, routing, frequent topology changes. A mobile ad hoc network can be deployed anywhere at any time therefore are used in situations such as during earthquake, floods, disasters etc [20].

It is due to unique art of an ah-hoc network formation routing has attained a primary focus. In essence routing indicates route establishment between two communicating devices in a network. In the existing literature several routing mechanism have proposed some of the known schemes [18, 19, 20, 22, and 27] are Destination Sequence Distance Vector (DSDV), Dynamic Source Routing (DSR), Ad-Hoc On-demand Distance Vector Routing (AODV), Temporally Ordered Routing Algorithm (TORA) and Mobile Ad-hoc on Demand Data Delivery Protocol (MAODDP). It is known that routing in ad-hoc network still needs some refinement towards more effective standard routing solution. In this context a sufficient amount of work has been conducted modifying TCP to support routing in ad-hoc network.

Transmission control protocol is the most reliable transport layer protocol for Internet. The major functions of TCP is end-to-end connection, congestion control, flow control, in order delivery of packets and reliable transportation of data packets and is performing well in wired networks with the above mentioned features, that’s why TCP is the backbone of internet [13]. It is a known fact that due to unique nature of mobile ad-hoc network packets are lost because of frequent path breaks due to mobility of destination node or mobility of the nodes working as routers between source node and destination node, high bit error rate in the wireless channel, collisions due to hidden terminals etc [9]. Many protocols have been developed in the context of TCP but none of them is capable enough to improve TCP performance over mobile ad-hoc network.

Node Feedback TCP based mechanism [49] aims to enhance TCP performance in a mobile ad-hoc network. One of the distinguishing features of NFBTCP is the feedback from the active node to TCP. After route failure, once the communication is re-established nodes which were in active communication before the communication or the route path broken is responsible to inform about the link capacity to the TCP. TCP adjust the size of congestion window (CWND) according to the link capacity of the established connection. In this way TCP doesn't need to invoke slow start mechanism. This feedback assists TCP in adjusting the size of congestion window; therefore TCP doesn't need reinitiating slow start mechanism. Moreover, this scheme also proposed solution to the out of delivery problem of TCP in mobile ad-hoc network.

NFBTCP uses notification failure to enable TCP differentiating between the real congestion and congestion assumed by the TCP due to link loss or route failure. We believe proposed mechanism will not only overcome some of the weaknesses of the existing schemes but also will yield an efficient TCP modified version for mobile ad-hoc network. NFBTCP has coded in java and evaluated in SWAN. A higher number of route requests and route replies representing networking activities were observed with the increase of mobile nodes. It can be well understood that increase of mobile nodes to some extent implies increase in the communication takes place in a network. Moreover to the messages activities, good numbers of routes were added at the end of each simulation cycle. It is quite understandable that the more routes available for data transfer in mobile ad-hoc network, the better. Furthermore, such additions to the available routes could directly impact overall throughput. It is due to the nature of mobile ad-hoc network, where routes forms and broken almost unexpectedly. Therefore an alternative route to the destination is always beneficial. Lastly, nodes in mobile ad-hoc network suffer with limited resources. That makes conservation of all such resources an important issue in mobile ad-hoc network. NFBTCP has shown satisfactory performance by conserving available memory in most of the conducted experiments. This chapter introduces research and problem domain of this project, which are discussed in due course and has been organised as follows. In section 1.2 Transmission Control Protocol is introduced followed by a discussion of its performance in mobile ad-hoc network in section 1.3. Aims and objectives are presented in section 1.4. In section 1.5 a summarised view of NFBTCP is covered and the organisation of this dissertation is presented in section 1.6

1.2. The Transmission Control Protocol

Transmission control protocol works on the transport layer of OSI model stack. The main functions of TCP are end-to-end connection, congestion control, flow control, in-order delivery of packets and reliable transportation of data packets [5]. Throughput of a network degrades if the transport layer protocol cannot perform the above-mentioned functions properly. TCP should maximize the throughput by differentiating between congestion and link failure and should take appropriate actions according to the problem occurred. If it's real congestion then it should inform the sender to slow down the sending rate of data packets. And if it's not congestion and the loss of packets is due to link failure then it should inform the sender about link failure and not to send data packets to the destination node. Congestion window controls the sending rate of data packet to the destination. The network throughput depends on the size of congestion window. The size of congestion window gets lager depending on the rate of arrival of every new acknowledgment received by the TCP sender. When the data packet is lost and the sender does not receive ACK from the receiver within the retransmission timeout period then TCP shrinks its congestion window and invokes congestion control mechanism [9].

A significant amount of research has been done to make TCP capable of supporting communication over mobile ad-hoc network [4, 5, 7, 8, 9, 10, 11, 12 and 13]. However, despite numerous attempts TCP failed to show impressive performance in such an environment. Although TCP provides reliable end-to-end delivery of data over wired networks, several recent studies have indicated that TCP performance degrades significantly in mobile ad hoc networks. This is mainly because TCP considers any packet loss and/or delay as a congestion signal although MANET encounters several types of losses and delays that are not related to congestion. Non-congestion losses/delays mainly occur because TCP cannot adapt well to such mobile wireless multi-hop networks. The following subsections discuss different factors that affect TCP performance in MANET.

In mobile ad-hoc network packets loss is quite frequent due to frequent path breaks as a result of mobility of destination node or mobility of the nodes working as routers between source node and destination node, high bit error rate in the wireless channel, collisions due to hidden terminals etc, when the data packet is lost and the sender dose not receive acknowledgement from the receiver within the retransmission timeout period then TCP assumes this as congestion and invokes the congestion control mechanism [9]. When TCP assumes packet loss as congestion then it shrinks its congestion window and reduces the packet transfer rate and thus degrades overall throughput of the network. To gain high throughput from the network TCP should differentiate between congestion and packet loss due to mobility or path breakage.

1.3. Problem statement.

To address the problems experienced by TCP in MANET, a number of proposals have been presented. The vast majority of these proposals are TCP modifications that address some particular TCP inefficiency. The main design requirement is indeed to keep the improved transport protocol backward compatible with the legacy TCP, so that "improved" and "legacy" users may be able to communicate with each other. The differences between MANET and traditional wired networks are so many, that TCP would need a large number of modifications to work in this environment. It is well known that TCP shows a number of different issues in mobile ad-hoc networks. These issues till-to-date pose an open question to the research community. Existing literature report various solution, however these solutions lack in one way or the other. Therefore the problem still stands as before, clearly there is a need of a solution which can cope with TCP issues in ad-hoc network. We stress that without taking into consideration typical ad-hoc network environment a successful solution is difficult to develop. Moreover, we understand the solution should be designed in a manner which can deliver solution of number of different problem in sequence they occur in mobile ad-hoc network environment. In the light of the above discussion we propose Node Feedback TCP based mechanism to enhanced TCP performance in mobile ad-hoc network.

1.4. Aims and Objectives

In view of the above discussion it can clearly be understood that TCP performance over mobile ad-hoc network has been a challenge and thus needed to be resolved. We have proposed and develop a novel scheme NFBTCP to enhance TCP effectiveness within an ad-hoc network. It was discovered that TCP performance in a mobile ad hoc network is degraded due to ad-hoc network formation and operational pattern. A set of aims and objectives were defined in between from the start to the end of this projects which are as follows.

- We have investigated TCP performance in mobile ad-hoc network in order to develop an understanding of the research domain at a wider level.
- We have evaluated existing TCP based routing solution for mobile ad-hoc network with a view of identifying weaknesses in the reported schemes.
- We have proposed a novel TCP based solution for mobile ad-hoc network. The proposed solution addresses identified known weaknesses as reported in the existing solutions.
- The proposed scheme was designed using standard design mechanisms so that it could be used to implement the scheme using a computer language. Moreover, this design was helpful in looking back towards any improvement during the implementation stage of NFBTCP.
- The developed scheme was evaluated in a simulation frame. Evaluation experiments were conducted to monitor and performance of the proposed scheme in varying simulation environments. It is important to mention that the scheme has showed an expected performance with satisfactory results are achieved at different experiment's cycle.

1.5. Research contribution.

Node feedback TCP based mechanism introduces a new flavor of TCP for mobile ad-hoc network. It follows an intermediate approach in between some of the existing mechanisms of TCP based schemes for mobile ad-hoc network. We have addressed TCP slow start mechanism in the context of mobile ad-hoc network and introduce measures through whom TCP can differentiate between real congestion and congestion assumed by TCP due to packet lost or route failure in mobile ad-hoc network. In addition our proposed mechanism also deals with out-of-order delivery problem of TCP in mobile ad-hoc network. It is important to mention that NFBTCP not only addresses TCP related issues but also provides a number of different operations to assists in the smooth running of an ad-hoc network.

The scheme has been developed in Java and evaluated using SWANS. Evaluation observations detail the confirmation of the theoretical concepts which were included as a part of the functional specification of NFBTCP. NFBTCP has validated itself as a fully functional and operational-able for mobile ad-hoc networks, thus should be taken as a new novel TCP based solution for mobile ad-hoc networks. In the light of the conducted experiments, it can be seen that NFBTCP performed well in all simulation environments. A higher number of route requests and route replies representing networking activities were observed with the increase of mobile nodes. This clearly showed that NFBTCP fits well within mobile ad-hoc networking environment.

Increase of mobile nodes to some extent implies increase in the communication taking place in a network. In addition to the message activities, good numbers of routes were added at the end of each simulation cycle. It is quite understandable that the more routes available for data transfer in mobile ad-hoc network, the better. In this context unlike traditional TCP better route connectivity is possible through NFBTCP. In addition to the above, more available routes could directly impact overall throughput. It is due to the nature of ad-hoc network, where routes forms and broken almost unexpectedly. Therefore an alternative route to the destination is always beneficial. It is a well-known fact that nodes in mobile ad-hoc network suffer with limited resources. That makes conservation of all such resources an important issue in the context of mobile ad-hoc network. NFBTCP has shown self-explanatory performance by conserving available memory in most of the experiments. It can be easily understood that a direct way of measuring the proposed scheme performance is to monitor the throughput. A particular attention was given to that where it was found that most of the simulation cycle ended with an expected throughput. Since TCP is used for transporting data packets, therefore having most of the sent packets delivered at the destination by NFBTCP implies that the scheme has met the desired end objective. We believe NFBTCP offers a complete and an effective TCP based routing solution for mobile ad-hoc networks.

1.6. Organization

We have followed a defined pattern from the start to the end of this research project. The whole problem is divided into sub tasks and each of these has been compiled in different chapters of this thesis. Introductions to each of these chapters are as follows.

Chapter 1. Introduction: An introduction to the problem area with a brief background is explained in this chapter. In addition, a summary of the developed solution for the chosen problem and the research contribution are also briefed.

Chapter 2. Literature Review: This chapter presents an in-depth explanation of the research domain, previously reported solutions and critical analysis with a view to understanding weaknesses within the reported architectures.

Chapter 3. Problem Analysis: In this chapter the identified problem has been analyzed in view of the related reported work.

Chapter 4. NFBTCP: This chapter focuses on the proposed scheme. In this context, an explanation of the associated functions and overall benefit of the developed scheme are also presented. NFBTCP offers a new novel TCP based solution for Mobile Ad-hoc Networks. It was necessary to have the structural design of the scheme conducted prior to implementation. This chapter presents the proposed scheme design, pseudo codes and implementation details.

Chapter 5. Evaluation and Discussion: NFBTCP has been evaluated in SWANS through different simulation pattern. The scheme has proved to be implementable and efficient to support communication operation in mobile ad-hoc network. Evaluation has been concluded in discussion section within this chapter.

Chapter 6. Conclusions and Future Work: This section presents conclusions derived both from background research and the proposed scheme development and evaluation phases. In addition, thoughts regarding future research directions are also presented.

This chapter focuses on the introduction of the problem being investigated besides general introduction of the problem domain in general. The following chapter analyzes some of the previously reported solution.

Chapter 2. Literature Review

2.1. Introduction

A Mobile Ad-hoc Network is a collection of mobile nodes connected together in the absence of any fixed infrastructure. Nodes in ad hoc networks work both as hosts and routers forwarding data packets for other nodes in the network [22]. This process may involve multiple intermediate nodes, and it may produce the establishing of a multihop connection (multi-hop ad hoc network) between sender and receiver. These networks are appropriate for scenarios where wired networks are not possible such as in a disaster recovery, battlefield, short-lived networks as in conference spots, etc. In the last few years MANET are emerged as a flexible and low-cost extension of wired infrastructure networks. MANETs inherit the traditional problems of wireless communication and wireless networking, like high bit error rate, high sensitivity of wireless channel from outside signals, the possibility of path asymmetry, and so on. In addition, the multihop nature of connections, the lack of a fixed infrastructure, and node mobility add new problems, such as network partitions, route failures, and the hidden terminal. These new problems pose a number of design constraints that are specific to ad-hoc networking.

2.2. OSI Reference Model

Protocol layering is a common technique to simplify networking designs by dividing them into functional layers, and assigning protocols to perform each layer's task [31]. For example, it is common to separate the functions of data delivery and connection management into separate layers, and therefore separate protocols. Thus, one protocol is designed to perform data delivery and another protocol layered above the first performs connection management. The data delivery protocol is fairly simple and knows nothing of connection management. The connection management protocol is also fairly simple, since it doesn't need to concern itself with data delivery. Protocol layering produces simple protocols, each with a few well-defined tasks.

There are seven layers in the OSI reference model. Open system interconnection is a standard for worldwide communications that defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer on source node and proceeding to the bottom physical layer and on the receiving node the control is passed from physical layer proceeding to the bottom application layer to complete transfer of data packets. In the wireless protocol stack there are five layers. The application layer performs the functions of presentation and session layers. A brief introduction to different layers is as follows.

- **APPLICATION LAYER:** The application layer interfaces directly to and performs common application services for the application processes. The common application services provide semantic conversion between associated application processes.
- **TRANSPORT LAYER:** The transport layer provides reliable transfer of data between end users. The transport layer can keep track of the packets and retransmit those that fail.
- **NETWORK LAYER:** The network layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination via one or more networks while maintaining the quality of service requested by the Transport layer. The Network layer performs network routing, flow control, segmentation/desegmentation, and error control functions.
- **DATA LINK LAYER:** The data link layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical layer.
- **PHYSICAL LAYER:** The physical layer defines all electrical and physical specifications for devices. This includes the layout of pins, voltages, and cable specifications. Hubs and repeaters are physical-layer devices.

2.3. Characteristics of Mobile Ad-Hoc Network

A mobile ad-hoc network can be thought of as a collection of mobile platforms each combining the functionality of a router and a host as shown in figure 2.1. These mobile platforms, also called hosts are attached to a number of wireless communication mediums and are free to move about arbitrarily. These hosts are equipped with wireless transmitters and receivers using antennas which may be Omni directional, highly directional, possible steerable or some combination thereof. A mobile ad-hoc network is established through the mutual co-operation of mobile nodes that forward and receive packets for each other. Mobile ad-hoc networks offers self-configuration and on the fly network facilities to places where it is not possible otherwise. Two or more nodes can form an ad-hoc network without need of a centralized infrastructure. In due course, routing is achieved through routing protocol finding distension of interest for nodes in a network. It is understandable that different protocols adopt various strategies to offer such services.

Node mobility introduces certain scalability problems in mobile ad-hoc network protocols, when the network topology changes frequently control messages have to be sent between nodes so that new routes are found and propagated through the network [20] as shown in figure 2.1A and 2.1B.

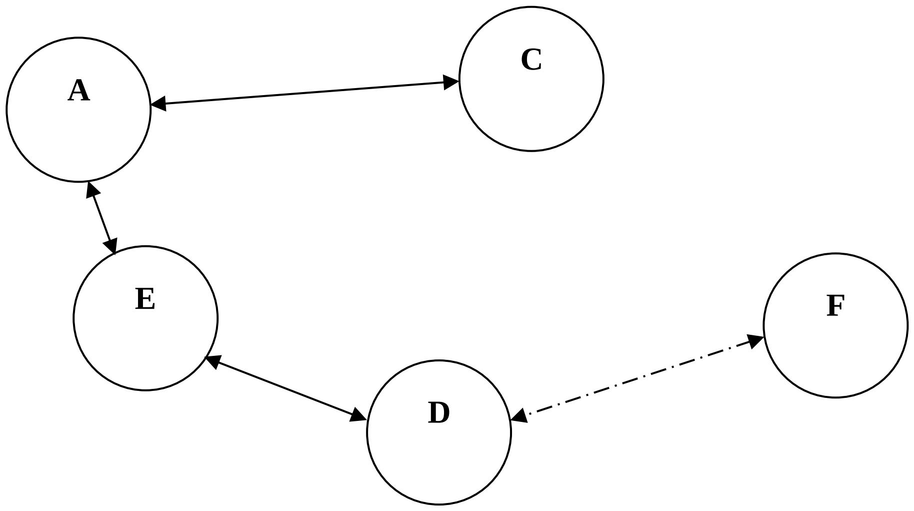


Figure 2.1A

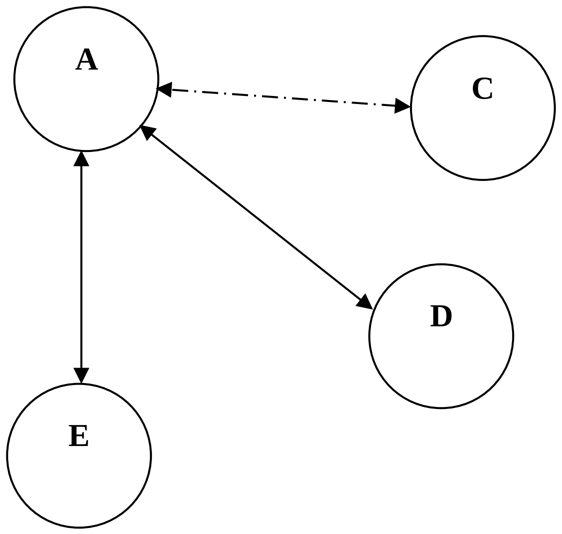


Figure 2.1B

In networks where the topology changes infrequently, it is reasonable to expect that when this happens there might be a short period where lots of control messages will propagate through the network to distribute the new destination paths. But when a high rate of topology change is one of the characteristics of the network, the protocol designer should make provision for highly dynamic and fast adapting algorithms that minimize control messages and attempt to utilize long-lived routes to the maximum extent.

In many kinds of mobile ad-hoc network, mobile nodes usually rely on exhaustible means for providing energy, such as batteries. For these nodes, energy conservation suddenly becomes an important design decision [26]. Nodes with low battery power may decide to enter a power saving mode when they having nothing to send or until another high priority event is generated. This behavior may affect the way the whole network is operating, since each node is responsible for forwarding other node's packets, apart from its own. If nodes decide to become "selfish" and break the collective and cooperative nature of ad hoc networking by not forwarding other node's data the ad hoc architecture is endangered. A multitude of other problems and design trade-offs concerned with power utilization in such networks and the particular area is becoming the focus of increased attention.

Last but not least we should examine briefly some security issues in ad-hoc networks. As with any wireless communications, ad-hoc networks can be highly vulnerable to security threats. On one hand, their distributed nature makes it difficult to implement any security scheme that relies on a central authority. While on the other hand there is an increased possibility for eavesdropping, denial-of-service and ma-in-the-middle attacks. Add to this the fact that the more secure architecture is the slower and more cumbersome the whole issue of security in ad hoc networks becomes somewhat problematic. Nevertheless, the decentralized architecture provides a more resilient approach to single points of failures.

2.3.1. Applications of Mobile Ad-hoc Networking

Some of the application of ad-hoc network is communication in situations where battlefield survivability counts or infrastructure is non-existent, which is the case during disaster relief or rescue operations [30]. Both of these applications rely on the decentralized and cooperative attributes of mobile ad-hoc networks. However a number of other applications can be envisioned.

- **Conferencing:** This scenario envisages a group of people gathering in the same area and exchanging shared information using the multi-hop characteristics of ad hoc network [26]. Currently, this is done by requiring everyone to connect to a central network which at times might be unavailable or the overhead might be too costly when all that is required is the sharing of small amount of data.
- **Ubiquitous Computing:** If projections and estimations are correct then soon we could be living in a world where electronic devices can join spontaneously in established networks and exchange data with other devices in their close vicinity, in a transparent and simple fashion. Bluetooth is an emerging standard for realizing such a vision and is backed by such companies as Ericsson Inc, IBM, Intel, Microsoft and Nokia [26]. It is a short-range radio technology aimed at eliminating wires between electronic devices. Bluetooth allows up to eight devices to be connected into what is called a piconet [28]. This could be a suitable technology that an ad-hoc network could use for transferring information without utilizing fixed infrastructure.
- **Data Gathering:** Another application where ad-hoc networks can prove useful is the collection of data from remote areas like air or sea [28]. A network of sensors can contain small, inexpensive, short-range radio transmitter that can collectively gather and forward data towards a base station. Current alternatives include either storing data for later collection or using large or expensive satellite transmitters.

But for cases like animal tracking where a small transmitter is essential, ad-hoc network can prove be to be a useful too, provided there is a reasonable coverage of the specific area with tracking devices.

There are also existing and future military networking requirements for robust, IP-compliant data services within mobile wireless communication networks many of these networks consist of highly-dynamic autonomous topology segments. Also, the developing technologies of "wearable" computing and communications may provide applications for mobile ad hoc networks. When properly combined with satellite-based information delivery, Mobile ad hoc networks can provide an extremely flexible method for establishing communications for fire, safety, rescue operations or other scenarios requiring rapidly deployable communications with survivable, efficient dynamic networking. There are likely other applications for MANET technology, which are not presently realized or envisioned by the author. It is simply put, improved IP-based networking technology for dynamic, autonomous wireless networks.

2.3.2. Problems, Constraints and Challenges of Mobile Ad hoc Networks

When designing mobile ad-hoc networks, several interesting and difficult problems arise due to the shared nature of the wireless medium, the limited transmission range of wireless devices, node mobility, and battery limitations. This section will describe some of these problems.

- **Dynamic topologies:** The network topology of an ad-hoc network is very dynamic as the mobility of nodes or memberships of nodes are very random and rapid [30]. This emphasizes the need for routing solutions to be dynamic.
- **Bandwidth-constrained:** variable capacity links: Wireless links will continue to have significantly lower capacity than their hardwired counterparts. In addition, the realized throughput of wireless communications after accounting for the effects of multiple access, fading, noise, and interference conditions, etc is often much less than a radio's maximum transmission rate [30]. One effect of the relatively low to moderate link capacities is that congestion is typically the norm rather than the exception, i.e. aggregate application demand will likely approach

or exceed Network capacity frequently. As the mobile network is often simply an extension of the fixed network infrastructure, mobile ad hoc users will demand similar services. These demands will continue to increase as multimedia computing and collaborative networking applications rise.

- **Energy-constrained operation:** Some or all of the nodes in a mobile ad-hoc network may rely on batteries or other exhaustible means for their energy [30]. For these nodes, the most important system design criteria for optimization may be energy conservation.
- **Limited physical security:** Mobile wireless networks are generally more prone to physical security threats than are fixed- cable nets [20]. The increased possibility of eavesdropping, spoofing, and denial-of-service attacks should be carefully considered. Existing link security techniques are often applied within wireless networks to reduce security threats. As a benefit, the decentralized nature of network control in mobile ad hoc network provides additional robustness against the single points of failure of more centralized approaches.

In Summary, these characteristics create a set of underlying assumptions and performance concerns for protocol design, which extends beyond those guiding the design of routing within the higher-speed, semi-static topology of the fixed Internet.

2.4. Routing Protocols for MANET

Development of routing protocols for ad hoc networks has been one of the hottest topics within this area in recent years. As a consequence, a large number of routing protocols have been designed, either by modifying Internet routing protocols, or proposing new routing approaches.

MANET routing protocols are typically subdivided into two main categories: proactive routing protocols and reactive on-demand routing protocols [19]. Proactive routing is derived from the traditional distance vector and link state protocols developed for the internet. The primary characteristic of proactive approaches is that each node in the network maintains a route to every other node in the network at all times. Route creation and maintenance is accomplished through some combination of periodic and event-triggered routing updates. This approach has the advantage that routes are available at the moment they are needed. A source can simply check its routing table, when it has data packets to send to some destination, and begin packet transmission. However, the primary disadvantage of these protocols is that the control overhead can be significantly large.

Reactive on demand routing protocols take a very different approach than proactive protocols, since they do not maintain a route between all pairs of network nodes. Instead, reactive protocols discover the route to a destination only when there is a demand for it. Specifically, when a source node needs to send date packets to some destination, it checks its routing table to determine whether it has a route. If no route exists, it performs a route discovery procedure to find a path to the destination. Hence, route discovery becomes on-demand. With this approach, if two nodes never need to talk to each other, then nodes in the network do not need to utilize their resources maintaining a path between each other.

The benefit of this approach is that signaling overhead is likely to be reduced compared to proactive approaches, particularly in networks with low to moderate traffic load. When the number of data sessions in the network becomes high, then the overhead generated by the reactive routing protocols may even surpass that of the proactive approaches. The drawback of reactive approaches is the introduction of route acquisition latency. That is, when a route is needed by a source node, there is some finite latency while the route is discovered. In contrast, with a proactive approach, routes are typically available at the moment they are needed.

2.5. Transmission Control Protocol

Transmission Control Protocol (TCP) is the standard for reliable connection-oriented transport protocols, and is normally used over IP (Internet Protocol) to provide end-to-end reliable communications to Internet applications [16]. TCP provides a reliable, connection-oriented, and full duplex type of service. In addition, TCP implements both flow control and congestion control mechanisms. The former prevents the TCP receiver’s buffer from being overflowed. The second is an end-to-end congestion control mechanism that prevents a process injecting into the network an excessive traffic load. Congestion control is concerned with the traffic inside the network. Its purpose is to prevent collapse inside the network when the traffic source (sender) is faster than the network in forwarding data. In a network with shared resources, where multiple senders compete for link bandwidth, it is necessary to adjust the data rate used by each sender in order not to overload the network. Packets that arrive at a router and cannot be forwarded are dropped, consequently an excessive amount of packets arriving at a network bottleneck leads to many packet drops. These dropped packets might already have travelled a long way in the network and thus consumed significant resources. Additionally, the lost packets often trigger retransmission, which means that even more packets are sent into the network. Thus network congestion can severely deteriorate network throughput. If no appropriate congestion control is performed this can lead to a congestion collapse of the network, where almost no data is successfully delivered [48].

2.6. TCP based routing for Mobile Ad-hoc Network

In the light of the conducted research it can be concluded that TCP based routing is an interesting and growing topic with in Ad-hoc networking. A good number of solutions concerning TCP in MANET have been reported. However these solutions suffer from certain weaknesses, thus requiring some effective mechanism to support routing operations [49]. In this section an overview of the existing schemes are presented. A summarize comparison of the studied schemes is shown in table 2.1

2.6.1. TCP-Feedback

TCP-Feedback uses a feedback based approach to avoid congestion in mobile ad-hoc networks [9]. It requires the support of a reliable link layer and a routing protocol that can provide feedback to the TCP sender about the path breaks. The routing protocol is expected to repair the broken path within a reasonable time period. When an intermediate node detects a path break, it originates a route failure notification (RFN) packet and sends it to the sender of a TCP session. The intermediate node that originates the RFN packet is called the failure point (FP). Every intermediate node that forwards the RFN packet understands the route failure, updates its routing table accordingly, and avoids forwarding any more packets on that route. If any of the intermediate nodes that receive RFN has an alternate route to the same destination, then it rejects the RFN packet and uses the alternate path for forwarding further data packets.

When a TCP sender receives an RFN packet, it goes into a snooze state. In the snooze state, a sender stops sending any more packets to the destination, cancels all the timers, freezes its congestion window, freezes the retransmission timer, and sets up a route failure timer. This route failure timer is dependent on the routing protocol and network size. When the break path rejoins or another path is detected then a route re-establishment notification (RRN) is sent to the sender and the sender changes from the snooze state to the connected state.

Critique

In the event of route failures, as the route re-establishment time increases, the use of feedback shows saving in unnecessary packet transmission. In TCP-F the RRN packet is generated when the intermediate node detects re-establishment of a broken path and it depends on information from the routing protocol. TCP-F has an additional state compared to the traditional TCP state machine, and hence its implementation requires modifications to the existing TCP libraries. Another disadvantage of TCP-F is that the congestion window used after a new route is obtained may not reflect the achievable transmission rate acceptable to the network and the TCP-F.

2.6.2. TCP with Explicit Link Failure Notification (TCP-ELFN)

TCP-ELFN uses explicit link failure notification for improving TCP performance in mobile ad-hoc network [13]. This is similar to TCP-F, except for the handling of explicit link failure notification (ELFN) and the use of TCP probe packets for detecting the route reestablishment. The ELFN is originated by the node detecting a path break upon detection of a link failure to the TCP sender. There are different ways in which the ELFN message can be implemented e.g. by sending an ICMP destination unreachable message to the sender. Once the TCP sender receives the ELFN packet, it disables its retransmission timers and enters into a standby state. In this state, it periodically originates probe packets to see if a new route is re-established. Upon reception of an ACK by the TCP receiver for the probe packets, it leaves the standby state, restores the retransmission timers, and continues to function as normal.

Critique

When a node detects a path break, it sends an ELFN packet to the sender about a broken path to stop further packets being sent to the destination, thus it can reduce congestion in the network. In TCP-ELFN when the network is temporarily partitioned, the path failure may last longer; this can lead to the origination of periodic probe packets consuming bandwidth and power. Another disadvantage is that the congestion window used after a new route is obtained may not reflect the achievable transmission rate acceptable to the network and the TCP receiver.

2.6.3. Ad-hoc Transmission Control Protocol (ATCP)

ATCP is implemented as a thin layer residing between the IP and TCP protocols and doesn't need changes in the existing TCP protocol [12]. The ATCP layer essentially makes use of the explicit congestion notification (ECN) for maintenance of the states. This layer is active only at the TCP sender. The major function of the ATCP layer is to monitor the packets sent and received by the TCP sender, the state of the TCP sender, and the state of the network. There are four states in the ATCP NORMAL, CONGESTED, LOSS, and DISCONN.

When a TCP connection is established, the ATCP sender is in NORMAL state. In this state, ATCP doesn't interfere with the operation of TCP and it remains invisible. When a packet is lost or arrives out-of-order at the destination, which generates duplicate ACKs. The ATCP sender counts the number of duplicate ACKs received and if it reaches three, instead of forwarding the duplicate ACKs to TCP, it puts TCP in a persist state and ATCP in the LOSS state. In the LOSS state, ATCP retransmits the unacknowledged segments from the TCP buffer. When a new ACK comes from the TCP receiver, it is forwarded to TCP and the TCP sender is removed from persist state and then the ATCP sender changes to the NORMAL state.

When the network gets congested, the ECN flag is set in the data and the ACK packets. When the ATCP sender receives this ECN message in the normal state, it changes to the CONGESTED state and just remains invisible, permitting TCP to invoke normal congestion control mechanism. When a route failure or network partition occurs in the network, ATCP expects the network layer to detect these and inform the ATCP sender through an ICMP destination unreachable message. Upon reception of the destination unreachable message, ATCP puts the TCP sender into the persist state and enters into the DISCONN state. It remains in the DISCONN state until it is connected and receives any data or duplicate ACKs. The connected status of the path can be detected by the acknowledgments for the periodic probe packets generated by the TCP sender. When ATCP puts TCP into the persist state, it sets the congestion window to one segment in order to make TCP probe for the new congestion window when the new route is available.

Critique

The advantage of ATCP is that standard TCP/IP is unmodified and it is invisible to TCP and therefore nodes with and without ATCP can interoperate. ATCP does not interfere with TCPs functioning in cases where the TCP connection is between a node in the wireless network and another in the mobile ad-hoc network. The drawback of ATCP is that nodes without ATCP will experience all of the performance problems associated with running TCP over a mobile ad-hoc network. In ATCP the congestion window is set to one segment which may not reflect the achievable transmission rate acceptable to the network and TCP receiver.

2.6.4. Split_TCP

Split-TCP splits transport layer objectives into congestion control and end-to-end reliability [8]. Split-TCP splits a long TCP connection into a set of short concatenated TCP connections called segments or zones, with a number of selected intermediate nodes known as proxy nodes. A proxy node receives the TCP packets, reads its contents, stores it in its local buffer, and sends an acknowledgment to the source (or the previous proxy) called local acknowledgment (LACK).

LACK does not guarantee end-to-end delivery. The responsibility of further delivery of packets is assigned to the proxy node. A proxy node clears a buffered packet once it receives LACK from the immediate successor proxy node for that packet. The source node clears the buffered packets only after receiving the end-to-end acknowledgment for those packets. Transmission control window at the TCP sender is also split into two windows, i.e. the congestion window and the end-to-end window. The congestion window changes according to the rate of arrival of LACKs from the next proxy node and the end-to-end window is updated based on the arrival of end-to-end ACKs.

Critique

The advantage of splitting a TCP connection into multiple segments is that once the packet makes it to a proxy it has traversed the previous segment and thus avoids having to travel all the way back to the source if the packet needs to be retransmitted. Split-TCP requires modifications to the TCP protocol structure. The overhead incurred in including frequent end-to-end ACKs in addition to the LACKs can consume extra bandwidth. The failure of proxy nodes or frequent path breaks affects the performance of split-TCP. The loss of end-to-end semantics may cause problems to applications that rely on such a guarantee provided by TCP.

2.6.5. Cross-Layer Approach

Cross-layer design is the interaction among the layers in the protocol stack [2]. For compatibility with the Internet, existing standard protocol stacks would be deployed in the new networks and mobile devices. However, these protocol stacks which are architected and implemented in a layered manner do-not function efficiently in mobile wireless environments. The system performance of future networks will be enhanced by cross-layer design between PHY, MAC and higher layer protocols [1]. Following is some of the key information available at different layers that can be exchanged among each other for cross-layer design.

- An application layer can communicate to other layers for the application's QoS needs, i.e. the delay tolerance, acceptable delay variation, required throughput and acceptable packet loss rate. TCP may provide packet loss and throughput information to the application. The application can use this input to adapt its sending rate.
- The information available with TCP is re-establishment time-out, congestion window, number of packets lost and actual throughput. [13]. When channel conditions are poor, retransmissions at the link layer result in delays which could lead to TCP retransmissions and

thus reduced throughput [9]. To avoid this, TCP and link layer could exchange retransmission information.

- The information available at the network layer is Mobile-IP hand-off initiation/completion events and the network interface currently in use. Mobile-IP hand-off delay may lead to reduced throughput due to the TCP retransmission time-out (RTO) and back-off mechanism. TCP can be informed about the event of Mobile-IP hand-off to reduce the retransmission latency.

2.6.6. Slow Start and Congestion Avoidance

A TCP sender must use the slow start and congestion avoidance algorithms to control the amount of outstanding data injected into the network [14]. To implement these algorithms, two state variables are added to the TCP per-connection state. The congestion window is a sender limitation on the amount of data the sender can transmit into the network before receiving an acknowledgment . The receiver's advertised window is a receiver limitation on the amount of outstanding data.

2.6.7. Fast Retransmit and Fast Recovery

When the TCP receiver receives an out-of-order segment, it sends an immediate duplicate acknowledgment to the TCP sender [13]. Duplicate acknowledgment happens when the TCP receiver receives an out-of-order segment and since it did not receive the segment(s) before this out-of-order segment, it cannot acknowledge the reception of this segment. Keeping in mind this fact, the TCP receiver responds with an ACK that has the sequence number of the expected packet, which is the same ACK it used to acknowledge the last in-order segment it received. This duplicate acknowledgment informs the TCP sender that the TCP receiver received an out of order packet and the sequence number of the expected packet. From the TCP sender's perspective duplicate acknowledgment can be caused by the following

· Dropped segments.

· The re-ordering of data segments by the network.

· Replication of ACK or data segments by the network.

2.6.8. Network Feedback Approaches

In these approaches, the network implements a monitoring mechanism that generates a notification message when it detects an abnormal event so that TCP may react [15]. TCP-F is proposed to overcome the TCP false reaction towards route failures in MANET. As soon as the network layer at any node detects the disruption of a route, it explicitly sends a Route Failure Notification packet to the source. Consequently, the TCP sender stops sending packets and freeze all its variables (such as timers and congestion window size). When one of the intermediate nodes learns about a new route to the destination, it sends a Route Re-establishment Notification packet to the source. The TCP sender leaves the snooze state, restarts the timers from their frozen values and resumes the transmission based on the stored sender window and timeout values. Similarly, the approach uses an Explicit Link failure Notification to inform the TCP sender about the route failure. The only difference from TCP-F is that the sender in the snooze state periodically probes the network and when an ACK is received, it considers it as an indication of route reestablishment.

A modified version of this approach is known as TCP-RC. TCP-RC recomputed the congestion window size and the slow start threshold for the TCP connection after the route is reconstructed instead of using the frozen values. An obvious limitation of this approach is that these techniques need to be deployed at every node. ATCP deals with the problems of high BER, route failures, network partitioning and multipath routing. A thin layer called ATCP is inserted between TCP and IP layers. The ATCP layer monitors TCP state and the state of the network (based on ECN and ICMP message) and takes appropriate action. The ATCP’s four possible states are: Normal, Congested, Loss and disconnected.

When ATCP sees that three duplicate ACKs have been received, it considers it a channel loss and only transmits the unacknowledged segments. Congestion is detected by ECN message. In case of temporary network partitioning, the ATCP receives an ICMP “Destination Unreachable” message. Hence, it puts the TCP sender in the persist state, sets TCP's congestion window into one and enters itself in the disconnected state. TCP periodically generates probe packets until it start receives their ACKs. This removes TCP from persist mode and moves ATCP back into normal state.

TCP-BuS (TCP Buffering capability and Sequence information) is another approach used to detect route failures. When a node detects a route failure, it sends an Explicit Route Disconnection Notification to the source containing the sequence number of the TCP segment pending in the head of the node's transmit queue. All the intermediate nodes will buffer the packets in their queues. When a route is discovered, the receiver sends to the sender the last sequence number it has successfully received. The sender only transmits the lost packets and the intermediate nodes starts sending the buffered packets.

A new approach called Split TCP to improve the performance of TCP in terms of fairness and throughput. This approach depends on splitting long TCP connections into shorter localized segments. The interfacing node between two localized segments is called proxy. The proxy intercepts TCP packets, buffers them and acknowledges their receipt to the source (or previous proxy) by sending a local acknowledgment. Upon the receipt of a LACK from the next proxy (or the final destination), a proxy will purge the packet from its buffer. The source keeps transmitting according to the rate of arrival of LACKs from the next proxy, but purges a packet from its buffer only upon receipt of an end-to-end ACK for that packet from the destination. This keeps the end-to-end reliability of TCP.

2.6.9. End-to-End Approaches

End-to-end approaches require no network support [16]. The end nodes (sender or receiver) can detect the network state by measuring appropriate traffic parameters. For example, high volume of out of order delivery signifies route change. A heuristic is employed to distinguish between route failures and congestion without relying on feedback from other network nodes. When timeouts occur consecutively, this is taken to be evidence of a route loss. The unacknowledged packet is retransmitted again but the RTO remains fixed until the route is reestablished and the retransmitted packet is acknowledged.

`TCP-DOOR (Detection of Out-Of-Order and Response) is another pure end-to-end approach to improve TCP performance by detecting and responding to out-of-order packet delivery events, which are interpreted as an indication of route failure. The non-decreasing property of ACK sequence numbers makes it simple for the sender to detect out-of-order delivery of non-duplicate ACK packets. To detect out-of-order delivery of duplicate ACK packets, they use one-byte TCP option which is incremented with each duplicate ACK packet.

Comparing the two approaches, we find that end-to-end approaches are easier to implement and provide more flexibility, while feedback approaches are more accurate as the information is coming directly from the network. Furthermore, it is clear that each approach deals only with one or a subset of the factors causing the bad performance of TCP in MANETs. However, most commonly, these solutions deal with route failures. Actually, this is reasonable because in such a dynamic environment the frequency of route failures is very high due to node mobility. We also find that most of the presented approaches take reactive actions. In these approaches TCP takes different actions rather than invoking congestion control when a non-congestion loss occurs. Some approaches are preventive (e.g. Split TCP). The target of this kind of approaches is to reduce the probability of other losses that may lead to false notification and unnecessary congestion control reaction.

2.6.10. TCP Variants

This section presents the main TCP variants that have been investigated in the literature. Each variant has its own features tailored to a specific problem faced by TCP congestion control, and in most cases each new variant represents an evolution of the previous one.

2.6.10a. TCP Tahoe

Tahoe represents the basic TCP version that was specified by Jacobson [13]. It was the first TCP designed to solve the congestion collapse affecting the Internet. Modern TCP implementations still use most of the mechanisms developed for Tahoe, as it will be shown below. In addition to the retransmit timeout mechanism, which was already implemented in early TCP-like transport protocols, TCP Tahoe counts on the three key mechanisms: Fast Retransmit, Slow Start, and Congestion Avoidance.

Critique

The Tahoe TCP implementation added a number of new algorithms and refinements to earlier implementations e.g. slow-start, congestion avoidance and fast retransmit. With Fast Retransmit, the data sender infers that a packet has been lost and retransmits the packet without waiting for a retransmission timer to expire, leading to higher channel utilization and connection throughput. Although Tahoe solved the congestion collapse problem, it rapidly proved to be too conservative by always resetting its CWND to one upon a lost packet.

2.6.10b. TCP Reno

TCP Reno conserved the three essential mechanisms of the basic TCP Tahoe, namely Slow Start, Congestion Avoidance and Fast Retransmit [15]. The novelty introduced into TCP Reno is the Fast Recovery mechanism. This mechanism prevents the communication path from going empty after Fast Retransmit, thereby avoiding the need to Slow Start to re-fill it after a single packet loss.

Fast Recovery is generally invoked when a TCP sender receives a predefined threshold of duplicate ACKs, just after the Fast Retransmit mechanism. This threshold, usually known as tcp rexmtthresh, is generally set to three. Once the threshold of dup ACKs is received, the sender retransmits the packet that seems to have been dropped and reduces its congestion window by one half. Unlike TCP Tahoe, TCP Reno does not invoke Slow Start, but uses the additional incoming duplicate ACKs to clock out subsequent outgoing data packets.

Fast Recovery assumes that each dup ACK received represents a single packet having left the pipe. Thus, during Fast Recovery the TCP sender is able to make intelligent estimates of the amount of outstanding data. Specifically, during Fast Recovery the usable TCP window is defined as min (rwin, cwnd+ ndup), where rwin refers to the receiver advertised window and ndup tracks the number of duplicate ACKs received. By using the ndup variable, the sender may estimate the amount of packets in flight. After entering Fast Recovery and retransmitting a single packet, the sender effectively waits until half a window of dup ACKs have been received, and then sends a new packet for each additional dup ACK that is received. Upon receipt of an ACK for new data (called a “recovery ACK”), the sender exits Fast Recovery by setting ndup to 0.

Critique

TCP Reno is optimized for the case when a single packet is dropped from a window of data. In such cases, the TCP sender can retransmit at most one dropped packet per Round-trip Time (RTT). TCP Reno is more efficient than its predecessor (Tahoe) but does not work so well when more than one packet is dropped from a window of data. The problem is that TCP Reno may reduce the CWND multiple times for recovering the lost packets, leading the connection to experience poor performance.

2.6.10c. TCP NewReno

NewReno improves the Reno implementation with regard to the Fast Recovery mechanism [44]. The objective of TCP NewReno is to prevent a TCP sender from reducing its congestion window multiple times in case several packets are dropped from a single window of data. NewReno can also avoid retransmission by timeout in scenarios where the involved congestion window is small preventing enough ACK packets from reaching the sender. In TCP Reno, when the sender receives a partial ACK packet it exits Fast Recovery. The term partial ACKs refers to ACK packets that acknowledges some but not all of the data packets that were outstanding when the Fast Recovery was started. Upon receipt of a partial ACK, the Reno sender brings the usable window back to the congestion window size, and so exits Fast Recovery. If there are sufficient outstanding packets, the sender may receive enough duplicate ACKs to retransmit the next lost packet (or packets) until all dropped packets are retransmitted by the Fast Recovery mechanism.
At every invocation of the Fast Recovery, CWND is halved. If there are not enough packets outstanding due to a low window size, then the sender needs to wait for the expiration of the retransmission timer. In this case the CWND is reset to one, inducing bandwidth wastage. Differently from Reno, the NewReno do not exit Fast Recovery when it receives partial ACKs. Instead, TCP NewReno treats partial ACKs received during Fast Recovery as an indication that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. Thus, when multiple packets are lost from a single window of data, TCP NewReno can recover without a retransmission timeout, retransmitting one lost packet per round-trip time until all of the lost packets from that window have been retransmitted. TCP NewReno remains in Fast Recovery until all of the data outstanding when Fast Recovery was initiated has been acknowledged. In this way, TCP NewReno avoids multiple reductions in the CWND or unnecessary retransmit timeout with Slow Start invocation, thereby improving the end-to-end performance.

Critique

TCP NewReno advances the Reno implementation with respect to the fast recovery mechanism by preventing a TCP sender from reducing its congestion window multiple times in case several packets are dropped from a single window of data TCP NewReno remains in fast recovery until all of the outstanding data is recovered without knowing bandwidth capacity can impact the overall throughput of the network.

2.6.10d. TCP SACK

Selective Acknowledgment preserves the basic principles of TCP Reno [13]. In fact, it uses the same algorithms of Reno for increasing and decreasing its congestion window. The novelty in TCP SACK lies in its behavior when multiple packets are dropped from one window of data, similarly to TCP NewReno. In SACK, the receiver uses the option fields of the TCP header (Sack option) for notifying the status of data received and queued by the receiver.

The SACK option field contains a number of SACK blocks, where each SACK block reports the received and queued bytes of data that are contiguous and isolated (there are gaps in the data stream). The first block in a SACK option is required to report the most recently received segment, and the additional SACK blocks repeat the most recently reported SACK blocks. The sender keeps a data structure called a scoreboard to keep track of the SACK options (blocks) received so far. In this way, the sender can infer whether there are missing packets at the receiver. If so, and if its congestion window permits, the sender retransmits the next packet from its list of missing packets.

In case there are no such packets at the receiver and the congestion window allows, the sender simply transmits a new packet. Like TCP Reno, the Sack implementation also enters Fast Recovery upon receipt of generally three duplicate acknowledgments. Then, its sender retransmits a packet and halves the congestion window. During Fast Recovery, SACK monitors the estimated number of packets outstanding in the path (transmitted but not yet acknowledged) by maintaining a variable called “pipe”. This variable determines if the sender may send a new packet or retransmit an old one.

The sender may only transmit if pipe is smaller than the congestion window. At every transmission or retransmission, pipe is incremented by one, and it is decremented by one when the sender receives a duplicate ACK packet containing a SACK option informing it that a new data packet has been received by the receiver. The Fast Recovery terminates when the sender receives an ACK acknowledging all data that were outstanding when Fast Recovery was entered. If the sender receives a partial ACK, i.e., an ACK that acknowledges some but not all outstanding data, it does not exit Fast Recovery.

For partial ACKs, the sender reduces pipe by two packets instead of one, which guarantees that a SACK sender never recovers more slowly than it would do if a Slow Start had been invoked. If it happens that a retransmitted packet is dropped, the SACK implementation reacts exactly as the Reno implementation. In such cases, the sender times out, retransmits and enters Slow Start.

Critique

SACK incorporates all the advantages found in NewReno and may recover multiple lost packets in a window of data in just one single RTT. A SACK implementation requires changes at both sender and receiver, though.

In similarity with the NFBTCP approach proposed here, SACK has the capability to distinguish between route congestion and route breakages .This allows the congestion window to be better controlled. However, SACK relies on a single ACK packet to determine that a route is still alive and resumes data flow on this link, but doesn’t utilize the full link capacity and therefore may have lower overall throughput than NFBTCP.

2.6.11. Explicit Congestion Notification

The ECN scheme specified in RFC 3168 [51] proposes to use network feedback to assist a TCP connection in reacting to congestion effects. By using this mechanism, TCP does not need to await a dropped packet due to buffer overflow to detect congestion and properly slow down. Rather, it is informed by the intermediate nodes (routers) when incipient congestion starts. ECN can prevent time wastage at the sender that, without ECN, always has to wait for either three duplicate acknowledgments or timeout timer expiration. The implementation of ECN requires specific flags in both IP and TCP headers. Two bits are used in each header for proper signaling among sender, routers and receiver. The active queue management (AQM) inside the routers marks packets when congestion reaches a given threshold. The receiver simply echoes back the congestion indication into the ACKs to the sender which reduces its sending rate to prevent severe congestion.

Critique

ECN is appealing for Internet use since it does not render any overhead regarding the current IP flows. Its drawback lies in the fact that to be effective, it requires changes to every network element. Other than this, it provides similar capabilities to the NFBTCP approach presented in this thesis in terms of link breakages, but doesn't cover other aspects such as link capacity and out-of-order delivery.

2.6.12. Delayed Acknowledgments (DA)

When data arrives at the receiver, the protocol requires that the receiver sends back an acknowledgment for reliability reasons [50]. The data packets are sequentially numbered so the receiver can acknowledge data by sending to the sender the sequence number of the highest data packet it has in its buffer. The acknowledgment scheme is cumulative, which means that by receiving the highest sequence number, the sender infers that all prior data were successful received. Thus, a TCP receiver does not necessarily have to transmit an acknowledgment for every incoming data packet.

RFC 813 [50] introduces a new mechanism that optimizes transmission efficiency by reducing the number of acknowledgments generated by a TCP receiver. This RFC shows that reducing the number of ACKs provides two benefits: lower processing overhead at the sender and robustness against the well-known Silly Window Syndrome (SWS). Measurements of TCP implementations, in particular on large operating systems, suggest that most of the overhead involved in a packet handling is not in the TCP or IP layer processing.

In fact, the most significant processing occurs in the scheduling of the handler that must deal with the packet at the sender. The delay ACK mechanism optimizes transmission efficiency by reducing the number of acknowledgments generated by a TCP receiver. However, if the network is facing constraints, additional mechanisms are needed to make sure that the receiver does not lead the sender to miss ACKs. Hence, RFC 813 recommends the use of a timer at the receiver to trigger ACK transmissions for data packets that do not arrive at the receiver in due time. This timer should be reset at every new incoming data packet and its duration could be either a fixed interval on the basis of the channel characteristics such as typical RTT or be adaptive to the channel conditions.

Critique

Although a delayed acknowledgment establishes the foundation for the delayed ACK mechanism, it does not specify clearly the actions to be taken by the receiver under a constrained channel. Again, its focus is on congestion rather than out-of-order delivery as tackled by NFBTCP.

TCP based Schemes	**End-to-End approach**	**Feedback approach**	**Fast Recovery**	**Slow Start**	**Congestion Avoidance**
TCP Tahoe	Yes	No	No	Yes	Yes
TCP NewReno	Yes	No	Yes	No	Yes
TCP-F	No	Yes	No	Yes	Yes
TCP-ELFN	No	Yes	Yes	No	Yes
ATCP	No	Yes	Yes	Yes	Yes
SPLIT TCP	Yes	Yes	No	Yes	Yes
TCP-DOOR	Yes	No	Yes	No	Yes

Table 2.1. Summary of TCP-based schemes for use in MANET.

2.7. Simulation tools.

SWANS was chosen for simulation experiments. SWANS capabilities are similar to existing simulators but is able to simulate much larger networks and have a number of other advantages over existing tools. SWANS can run existing Java network applications, such as web servers and peer-to-peer applications, over the simulated network without modification. The application is automatically transformed to use simulated sockets and into a continuation-passing style [40]. The original network applications are run within the same process as SWANS, which increases scalability by eliminating the considerable overhead of process-based isolation. Network packets in SWANS are modeled as immutable objects, allowing a single copy to be shared across multiple nodes. This saves the memory and time of multiple packet copies on every transmission.

In SWANS, simulation events among the various entities such as packet transmissions are performed with no memory copy and no context switch. The system also continuously profiles running simulations and dynamically performs code in lining, constant propagation and other important optimizations, even across entity boundaries. This is important, because many stable simulation parameters are not known until the simulation is running. Memory is critical for simulation scalability. Automatic garbage collection of events and entity state in SWANS not only improves robustness of long-running simulations by preventing memory leaks, it also saves memory by facilitating more sophisticated memory protocols. An example of memory savings in SWANS is the use of soft references for storing cached computations, such as routing tables. These routing tables can be automatically collected, as necessary, to free up memory. In the light of the above it is cleared that SWANS closely match the requirements of this research project. Thus it is selected as a simulation tool for evaluation of NFBTCP.

2.8. Summary

This chapter is an effort to analyze some of the previously proposed scheme of this area. In this context, introduction to the TCP along with a detail description of various TCP related segments followed by a detail description of various schemes. Previously presented solution could be categorized into one of two types. One type is the protocol schemes which are not an extension of TCP; whereas the other type is the schemes which are classified as TCP extensions. Most of the schemes discussed in this chapter focused on some specific issue rather than taking other interrelated issues. Moreover, presence of so many solutions to some extent emphasizes that the problem is still unresolved. The focus of the next chapter is to analyze identified problem in view of the related reported work.

Chapter 3. Research Methodology

3.1. Introduction

Taking into account TCP in mobile ad-hoc networks, many modified schemes are reported in the available literature. It is a well-known fact that these schemes do not fully address some of the main issues. The purpose of this chapter is to have an in-depth look into the issues revolved around TCP degrading performance in mobile ad-hoc networks. This will lead towards a conclusive problem analysis and its relation with potential solution to enhance TCP performance. In essence, this chapter will follow a prescribed sequence from basic to advanced operational pattern of TCP and its application over mobile ad-hoc network.

3.2. TCP in Mobile Ad-hoc network

The Transmission Control Protocol is one of the most authentic transport layer protocols for the Internet [17]. The most important functions of TCP are end-to-end connection, congestion control, flow control, in-order delivery of packets and reliable transportation of data packets. TCP performance has always been impressive in wired network and over the Internet [13]. A significant amount of research has been done to make TCP capable of supporting communication over mobile ad-hoc networks [4, 5, 6, 9, and 13]. However, despite numerous attempts TCP failed to show impressive performance in such an environment. Although TCP provides reliable end-to-end delivery of data over wired networks, several recent studies have indicated that TCP performance degrades significantly in mobile ad hoc networks. This is mainly because TCP considers any packet loss and/or delay as a congestion signal although MANET encounters several types of losses and delays that are not related to congestion. Non-congestion losses/delays mainly occur because TCP cannot adapt well to such mobile wireless multi-hop networks. The following subsections discuss different factors that affect TCP performance in MANET.

3.3. Factors Affecting TCP Performance in MANET

In addition to the traditional problems of wireless networking, the mobile multi-hop ad-hoc environment brings more challenges to TCP. In this section, we present a detailed analysis of all the factors that cause degradation in the performance of TCP in MANET.

3.3.1. TCP Congestion Control in Mobile Ad-hoc network

Congestion control is concerned with the traffic inside the network. Its purpose is to prevent collapse inside the network when the traffic source (sender) is faster than the network in forwarding data. To this end, the TCP sender uses a limiting window called congestion window. Assuming that the receiver is not limiting the sender, CWND defines the amount of data the sender can send into the network before an ACK is received. CWND controls the sending rate of data packet to the destination. The network throughput depends on the size of the congestion window. The size of the congestion window gets lager depending on the rate of arrival of every new ACK received by the TCP sender. When a data packet is lost and the sender dose not receives ACK from the receiver within the retransmission timeout period then TCP shrinks its congestion window and invokes its congestion control mechanism [9]. Throughput of a network degrades if the transport layer protocol cannot perform the above-mentioned functions properly. TCP should maximize the throughput by differentiating between congestion and link failure and should take appropriate actions according to the problem occurred. If it's real congestion then it should inform the sender to slow down the sending rate of data packets. And if it's not congestion and the loss of packets is due to link failure then it should inform the sender about link failure and not to send data packets to the destination node.

3.3.2. Network Partitioning

Network partitioning degrades throughput of TCP in mobile ad-hoc networks and is due to randomly moving nodes. The path will break if the sender and the receiver of a TCP connection lie in different partitions or any of the nodes between sender and receiver moves to another network. Due to path loss packets will be lost and TCP will assume it as congestion and will invoke the congestion control mechanism. Frequent disconnections cause a condition called serial timeouts at the TCP sender. This may lead to long idle periods during which the network is connected again, but TCP is still in the back off state and TCP will assume it as congestion and will invoke the congestion control mechanism.

3.3.3. TCP Connection Management

TCP is a connection oriented transport protocol. This means that one application process cannot send data to another; the two processes must first perform a handshake to open a TCP connection with each other. During the TCP connection establishment, both sides of the connection will initialize many TCP “state variables” associated with the TCP connection. The connection state resides entirely in the two end systems. The intermediate network elements do no maintain TCP connection state. A TCP connection provides for full duplex data transfer. If there is a TCP connection between process A and B, the application-level data can flow from A to B and from B to A at the same time. A TCP connection is also always point-to-point, that is, between a single sender and a single receiver. Multicasting is not possible with TCP.

3.3.4. Route Failures

In mobile ad-hoc networks packets loss is quite regular due to frequent path breaks caused by the mobility of destination nodes or mobility of the nodes working as routers between the source node and destination nodes, collisions due to hidden terminals etc, when the data packet is lost and the sender dose not receive acknowledgement from the receiver within the retransmission timeout period then TCP assumes this as congestion and invokes the congestion control mechanism [9]. When TCP assumes packet loss as congestion then it shrinks its congestion window and reduces the packet transfer rate and thus degrades overall throughput of the network. To gain high throughput from the network TCP should differentiate between congestion and packet loss due to mobility or path breakage.

3.4 Discussion

The focus of this section is to discuss shortcomings of TCP in MANET with the intention of addressing them within the proposed scheme in the next chapter. Transmission Control Protocol provides a reliable, connection-oriented and full duplex type of service. The major functions of TCP are end-to-end connection, congestion control, flow control, in-order delivery of packets and reliable transportation of data packets. Throughput of a network degrades if the transport layer protocol cannot perform the above-mentioned functions properly. TCP performs well in wired network and is considered as the backbone of the Internet [13]. In addition to the traditional problems of wireless networking, the mobile multihop ad hoc environment brings more challenges to TCP. In MANET, it may manifest in several forms like bandwidth asymmetry, loss rate asymmetry and route asymmetry. If the ACKs get bunched up, the sender may transmit data in a burst, which could lead to packet loss on the forward path. Also, disruption of the ACK stream can disrupt window growth and degrade performance to a fraction of the available bandwidth.

The main cause of route failures is node mobility. The route reestablishment duration depends on the underlying routing protocol, mobility patterns of nodes and traffic characteristics. It is possible that discovering a new route may take significantly longer than the RTO at the sender. As a result, the TCP sender will unnecessary invoke congestion control. If the sender and the receiver of a TCP connection lie in different partitions, all the sender's packets get dropped by the network resulting in the sender invoking congestion control.. In MANET, since the routes change many times during the lifetime of a TCP connection, the relationship between the congestion window size and the tolerable data rate becomes too loose.

In [7], the authors show that if the congestion window size is greater than an upper bound, the TCP performance will degrade. It is reported that, given a specific network topology and flow patterns, there exists an optimal TCP window size by which TCP achieves the best throughput. Unfortunately, TCP operates at an average window size, thus leads to increased packet loss due to the contention on the wireless channel. Since batteries carried by each mobile node have limited power supply, their life time is limited. Since each node acts as a router as well as an end system, unnecessary retransmissions of TCP segments consume this scarce power resource causing inefficient utilization of available power. Some routing protocols maintain multiple routes between source and destination to minimize the frequency of route re-computation. Unfortunately, this sometimes results in a significant number of out-of-sequence packets arriving at the receiver causing the generation of duplicate ACKs which causes the sender to invoke congestion control.

3.5. Methodology

We have followed a systematic approach from the start until the end of this project as shown in figure 3.1. Initially requirement analysis was done to understand the problem and associated task. The next step was to propose a suitable solution to address the identified problem. This solution was later on draw using standard design techniques. The same design was helpful during implementation process of the proposed solution. Implemented model was evaluated through various simulation cycles in order to monitor scheme performance in a practical environment.

The research methodology that I will adopt is quantitative approach. The advantage of the quantitative approach is that it measures, analyses and test the data and hence facilitates to test the hypothesis.

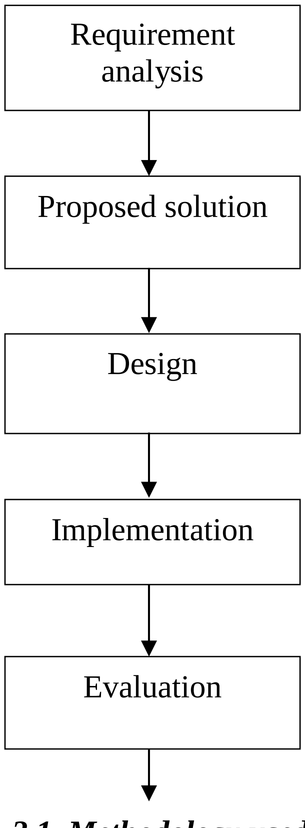


Figure. 3.1. Methodology used by NFBTCP.

3.6. Summary

In this chapter an in-depth look into the issues revolved around TCP degrading performance in mobile ad-hoc networks is presented. This has led towards a conclusive problem analysis and its relation with potential solution to enhance TCP performance. In essence, this chapter has followed a prescribed sequence from basic to advanced operational pattern of TCP and its application over mobile ad-hoc network. In addition a brief explanation of the selected methodology is also presented. In the following chapter Node Feedback Based Mechanisms is introduced.

Chapter 4. Node Feedback TCP Based Mechanism for Mobile Ad-hoc Network

4.1. Introduction

Node feedback TCP based mechanism introduces a new flavor of TCP for mobile ad-hoc networks. It follows an intermediate approach in between some of the existing mechanisms of TCP based schemes for mobile ad-hoc networks. We have addressed TCP slow start mechanism in the context of mobile ad-hoc networks and introduced measures through whom TCP can differentiate between real congestion and congestion assumed by TCP due to packet lost or route failure in mobile ad-hoc networks. In addition our proposed mechanisms also deal with the out-of-order delivery problem of TCP in mobile ad-hoc networks. It is important to mention that NFBTCP not only addresses TCP related issues but also provides a number of different operations to assists in the smooth running of ad-hoc networks. Otherwise, it was clearly difficult to have a clear understanding of NFBTCP modification of TCP for mobile ad-hoc network. In order to provide a concise and stepwise sequence of overall operation we give a clear overview from the start of the network until the over running of an ad-hoc network in congestion with TCP.

In the light of the background research of this thesis, it is well understood that TCP poor performance over ad-hoc networks is related to the typical nature of the ad-hoc network. Therefore it was necessary to define some of the interrelated operations alongside modifications to TCP. All of these operations are made part of this specification. Overall the operational structure of the proposed scheme can be seen in figure 4.1. In this context the overall specification of the proposed scheme will have all types of operation bundled within the same scheme. This is also done so as to support any future modification and to enable further development of the scheme. This chapter has been organized as follows: in section 2 specification of NFBTCP is given and a chapter summary is given in section 3.

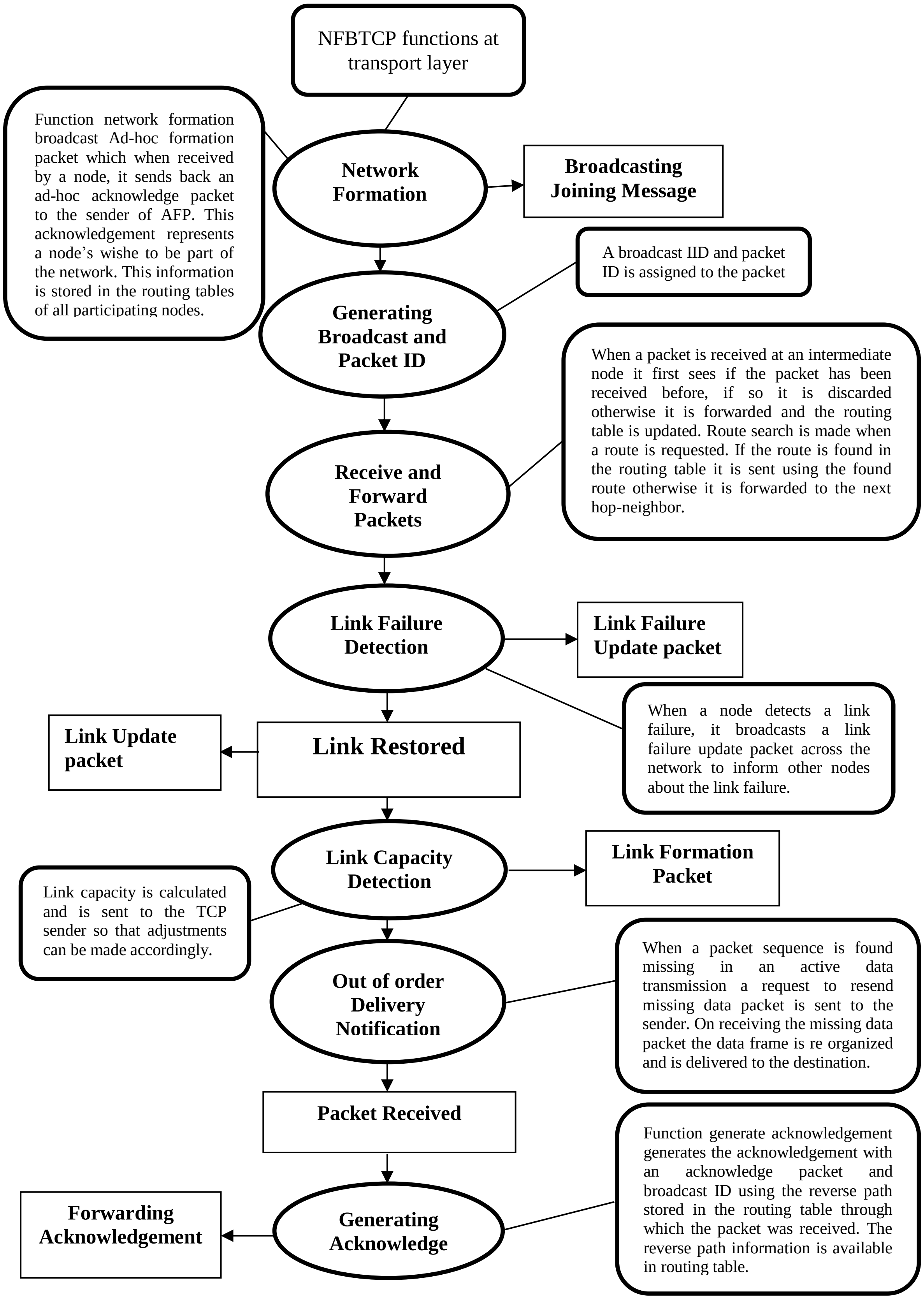

NFBTCP functions at transport layer
Function network formation broadcast Ad-hoc formation packet which when received by a node, it sends back an ad-hoc acknowledge packet to the sender of AFP. This acknowledgement represents a node's wishe to be part of the network. This information is stored in the routing tables of all participating nodes.
Network Formation
Broadcasting Joining Message
A broadcast IID and packet ID is assigned to the packet
Generating Broadcast and Packet ID
When a packet is received at an intermediate node it first sees if the packet has been received before, if so it is discarded otherwise it is forwarded and the routing table is updated. Route search is made when a route is requested. If the route is found in the routing table it is sent using the found route otherwise it is forwarded to the next hop-neighbor.
Receive and Forward Packets
Link Failure Detection
Link Failure Update packet
When a node detects a link failure, it broadcasts a link failure update packet across the network to inform other nodes about the link failure.
Link Update packet
Link Restored
Link Capacity Detection
Link Formation Packet
Link capacity is calculated and is sent to the TCP sender so that adjustments can be made accordingly.
Out of order Delivery Notification
When a packet sequence is found missing in an active data transmission a request to resend missing data packet is sent to the sender. On receiving the missing data packet the data frame is re organized and is delivered to the destination.
Packet Received
Function generate acknowledgement generates the acknowledgement with an acknowledge packet and broadcast ID using the reverse path stored in the routing table through which the packet was received. The reverse path information is available in routing table.
Forwarding Acknowledgement
Generating Acknowledge

Figure 4.1. Structural Representation of NFBTCP

The starting point of our implemented scheme is network formation, where a node broadcasts ad-formation packet to other nodes about establishment of an ad-hoc mobile network. Once an ad-hoc mobile is established, relevant information about all the nodes are stored in routing tables of all mobile nodes. When an intermediate node detects broken route informs other nodes about route failure. Similarly if alternative route is available it forward packets using that route. To avoid congestion due to path break in NFTCP sender is informed about broken path thus the sender stop sending further packets until new route is available. For link capacity detection link bandwidth is calculated after establishing new route to avoid invoking slow start mechanism of TCP. NFBTCP also deals with out-of-order delivery issue in TCP by storing packets of an active communication in a buffer at destination node and sends single ACK to the sender once the full message is received. Below is the summarized pseudo code of NFBTCP. The individual functions referred to in this pseudo-code will be discussed in more detail in the proceeding sections.

```
// When initialising the network
Fn network formation

// In the packet receive loop
When packet received
     If destination node
          Fn out-or-order packet delivery
               If packet missing or out-of-order
                    Request packet
               Else send single acknowledgment
               Endif
     Else receive and forward
          If broken link detected
               If no route to destination
                    Send link failure packet to sender
               Else route restored broadcast link update packet
```

```
                If new route found
                     Calculate link capacity
                     Broadcast link formation packet
                EndIf
           EndIf
      Endif
```

4.2. NFBTCP Specification and Operational Details

This section presents the NFBTCP specification which is a combination of the existing TCP functions and the functions defined by the proposed scheme. In this context, it is important to mention that the purpose of this section is to take into consideration all those aspects which are involved in the routine network operations. In the following section details of various functions alongside some of the associated operations of the main function are presented besides explanation of terms which are used as a part of this specification.

4.2.1. Network Formation

NFBTCP defines ad-hoc network formation in between two or more mobile nodes. In order to establish a network, a packet named as ad-hoc formation packet is broadcast by any node as shown in figure 4.2.1A. In figure 4.2.1A Node A is broadcasting ad-hoc formation packet to the rest of the nodes to form the potential network.

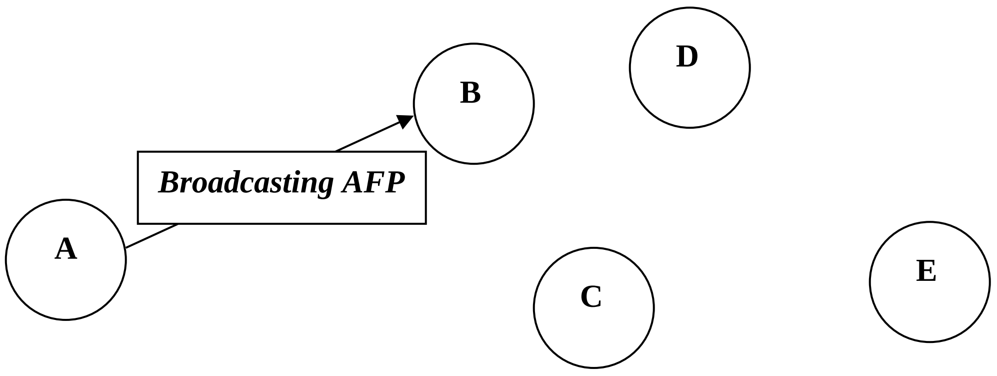


Figure 4.2.1A Node A broadcasting AFP

Nodes wish to be part of the network on reception of AFP send an Ad-hoc Acknowledge Packet back to the sender node as shown in figure 4.2.1B where the nodes apart from node A sending ad-hoc acknowledge packet. This packet shows nodes are agreed to join the network. This automatically updates all the participating nodes with the relevant location information about other nodes of the network.

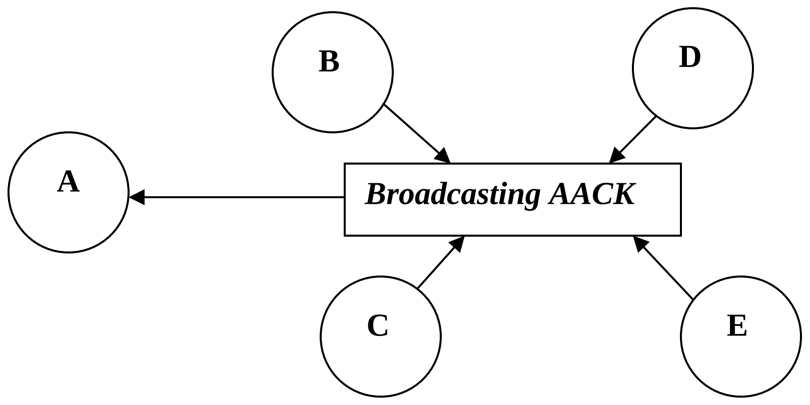


Figure 4.2.1B. Node broadcasting AACK

```
Function Network Formation

  Broadcast ad-hoc formation packet
  Received ad-hoc formation packet
  Initializes routing table
  Updates Routing Table
  Send ad-hoc acknowledgement

End Function
```

Function network formation broadcast Ad-hoc formation packet which when received by a node, it sends back an ad-hoc acknowledge packet to the sender of AFP. These acknowledge represents node wishes to be part of the network. This information is stored in routing table of all participating nodes.

It is important to note that this initial communication is taken as a starting point of communication between the participating nodes of a mobile ad-hoc network. These initial packets transmission is used to gather relevant information of other nodes of the network as shown in figure 4.2.1C where nodes are updating their self with the information of other nodes.

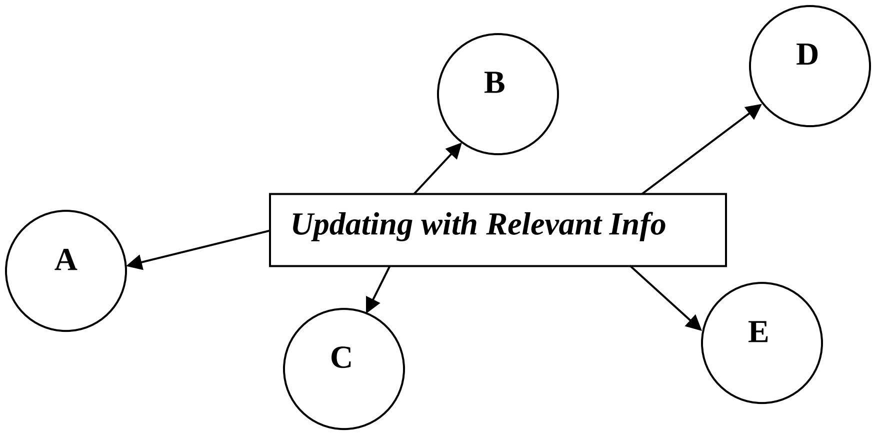


Figure 4.2.1C. Node Updating with Relevant Information of other nodes

NFBTCP besides introducing a unique way of network formation also enables any other nodes want to join the established network via ***broadcasting joining message***. Any node who was not part of the network at the time of network formation can broadcast joining message containing Joining Packet to introduce itself as a new participating node as shown in figure 4.2.1D.

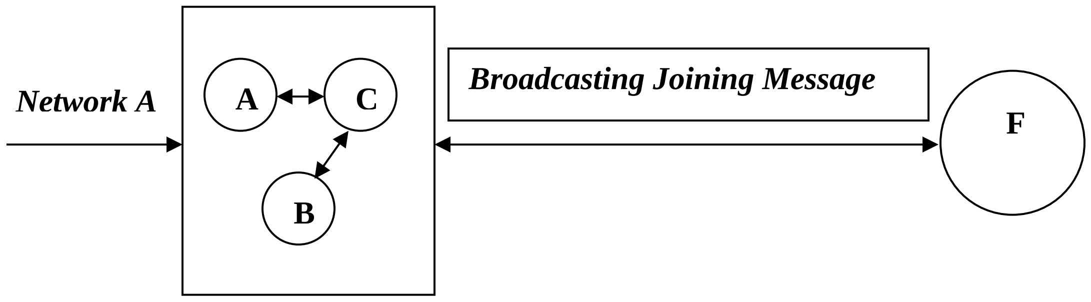


Figure 4.2.1.D. Node F broadcasting Joining Message

Below mentioned is the explanation of some the packets which are used within network formation of NFBTCP. Each of these packets are used to keep updated nodes with the current situation of the network as any node which receive any of these packet is required to forward it to the next hop neighbor. In this manner routes are formed and nodes in an ad-hoc network are linked with each other.

Ad-hoc Formation Packet (AFP). Ad-hoc formation packet is the packet which is broadcast when two or more mobile nodes want to form an ad-hoc network. This packet serves two purposes. Firstly, it indicates to the other nodes within the proximity that an ad-hoc network is about to establish and lastly it gives other potential nodes a starting point of communication via some relevant information about the other participating nodes. This packet also helps nodes in determining the hop-count of one node to the other node. Figure 4.2.1E describes AFP packet structure containing source sequence number and broadcast ID.

Source Seq no:	Broadcast ID	Test Data

Figure 4.2.1E AFP packet structure

Ad-hoc Acknowledge Packet (AACK). This packet is broadcasted by the potential nodes upon reception of ad-hoc formation packet. It serves two purposes; it shows willingness of a node or nodes to become one of the participants of the network and it also gives option to re-verify existing information about other nodes of the network. Ad-hoc Acknowledgement packet contains source sequence number, Broadcast ID, destination sequence number and acknowledgement of the received AFP packet as shown in figure 4.2.1F. The source sequence number, broadcast ID and destination sequence number are generated by the node in an incrementing manner.

Source seq no:	Broadcast ID	Destination Seq no:	ACK

Figure 4.2.1F AACK packet structure

Joining Ad-hoc Packet (JAP). Mobile nodes which were not the part of network at the time of network formation are require to send a Joining Ad-hoc Packet (JAP) to join the network. This packet serves two purposes i.e. informing participating nodes about the new joining nodes and to update and re-verify previously stored links. Joining Ad-hoc Packet contains source sequence number, Broadcast ID, destination sequence number and message data as shown in figure 4.2.1G.

Source seq no:	Broadcast ID	Destination Seq: no	Message data

Figure 4.2.1G JAP packet structure

4.2.2. Generating Acknowledgement

NFBTCP offer modification to the TCP scheme of generation acknowledgement for the sender nodes of a packet. When a packet is arrived at a destination it issues an acknowledgement back to the sender using the reverse path formed during the initial data transmission. The reverse path information is stored in the routing tables of all participating nodes. This process is known as generating acknowledgement and this is achieved via acknowledge packet delivery from destination to the source node of a packet. It should be noted that such packets are only sent when packets with data are received at some destination. However, for control packets their individual acknowledgement depends on the type of the packet is sent, some of such control packets and their acknowledgment types are defined in section 4.2.1. These packets are sent back to the source node using the same path developed during the delivery of packet from the source to the destination node. This whole operation could be viewed in figure 4.2.2. Where an acknowledged was send by Node B on receiving packet from Node A.

NFBTCP modifies TCP approach to discovered link break in an active path which is explained in the later section of this chapter. NFBTCP defines number of operations which are linked within the generating acknowledgement as shown in figure 4.2.2A. The details of these operations are discussed within this section and are as follows.

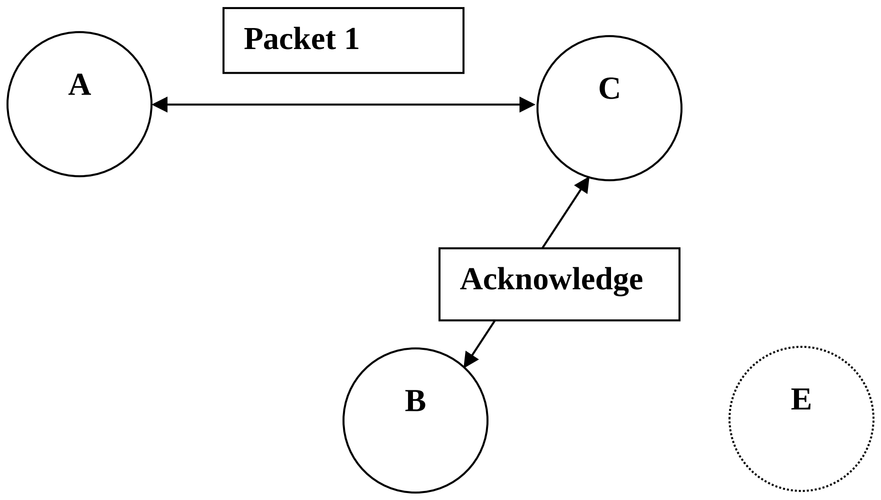


Figure 4.2.2. Node B is sending acknowledge packet to Node A via C

```
Function Generating Acknowledgement

  Generate Ack Packet ID
  Generate Ack Broadcast ID
  Include the Reverse Path

End Function
```

Function generate acknowledgement generates the acknowledgement with an acknowledge packet and broadcast ID using the reverse path stored in the routing table through which the packet was received. The reverse path information is available in the routing table.

Verifying Broadcast and Packet ID. An intermediate node of acknowledged packet performs three operations in sequence, verifying broadcast and packet ID being the first function during which intermediate or acknowledge packet receiving node verifies that it has not received the same packet before. If a packet with the same broadcast and packet ID has received before it is discarded and no further action is taken.

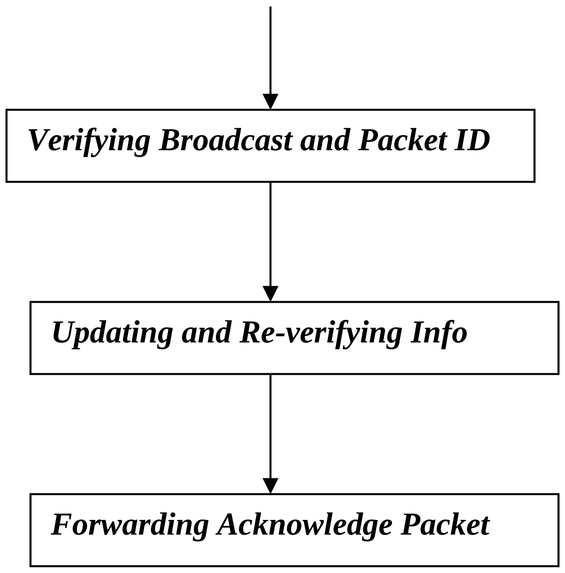

Verifying Broadcast and Packet ID
Updating and Re-verifying Info
Forwarding Acknowledge Packet

Figure 4.2.2A. Operations of intermediate nodes on receiving acknowledged packet.

Updating and Re-verifying Information. Once the freshness of received packet is confirmed the receiver node updates relevant information and forwards it for destination node or issue an acknowledgment packet if it is destination of the packet.

Forwarding Acknowledge Packet. Intermediate nodes are responsible to forward packets to the other node. In case of an acknowledged packet, the same route through which packet received from the sender route is used. Therefore nodes which are used during the first route use the same route from their storage to send it back to the source node. In figure 4.2.2.B Node C forwarding acknowledge packet from Node E to Node A

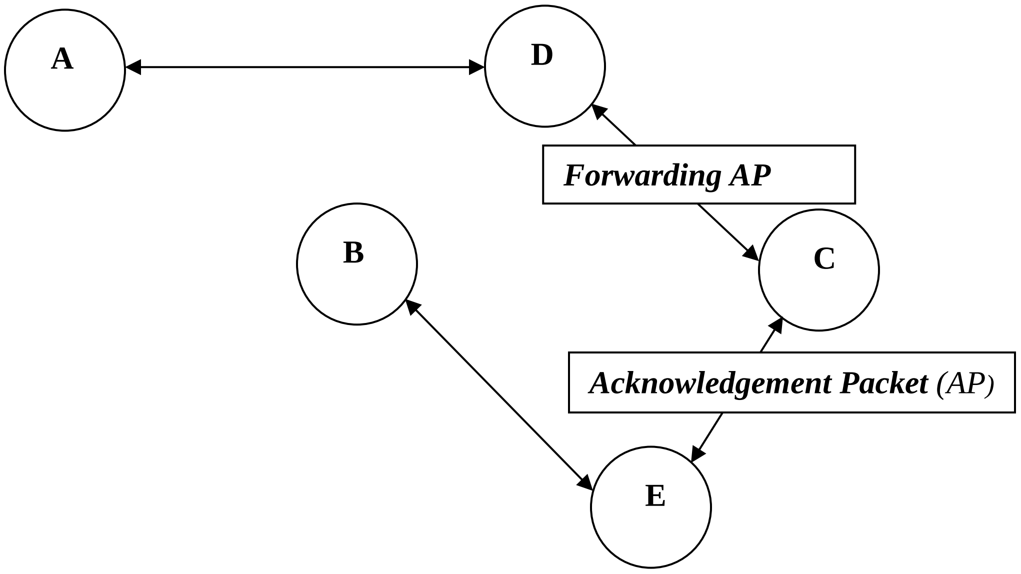


Figure 4.2.2B Node C Forwarding Acknowledge Packet from E to A

Acknowledgement Packet (AP). This packet is send by the destination node to the sender node. Acknowledgement Packet contains source sequence number, broadcast ID, destination sequence number and message data as shown in figure 4.2.2C.

Source Seq no:	Broadcast ID	Destination Seq no:	Message data

Figure 4.2.2C AP packet structure

4.2.3. Generating Broadcast and Packet ID

This is an important aspect of NFBTCP as this is requiring avoiding loop problem within an ad-hoc network. In general, if a packet is broadcasted and is not received at the destined location, it is possible that this packet will loop around the network. In this case either packet is eventually dropped or expired after its expiring time. Therefore in NFBTCP all the participating nodes are required to generate fresh broadcast and packet ID for each individual transmission as shown in figure 4.2.3. This same procedure is followed in some of the earlier mentioned control packets for different purposes.

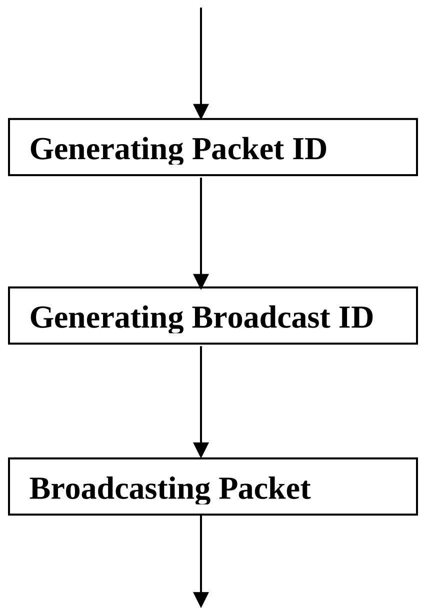


***Figure 4.2.3. Node Generating Packet and Broadcast ID*.**

```
Function Receive and Forward

  Received Packet
  Read Packet
  Search Routing Table for Received Packet ID and Broadcast ID
  IF Packet ID and Broadcast ID found
  Discard
  ELSE
  Update Routing Table
  Search for Route
  IF Route Found
  Forward
  ELSE
  Forward to next hop neighbor

End Function
```

When a packet is received at an intermediate node it first sees if the packet has been received before, in case if it is received before it is discarded otherwise it is forwarded and the routing table is updated.
Route search is made when a route is requested. If the route is found in the routing table it is sent using the found route otherwise it is forwarded to the next hop-neighbor.

4.2.4. Forwarding Data Packets

In NFBTCP whenever any intermediate node receives a packet destined for any other node of the network. It first see whether it has a route to the destination. If a route is found it uses the same route to transfer the received packet to the destination node. However, if no route is found for the destination node it forwards the data packet to the next hop neighbor.

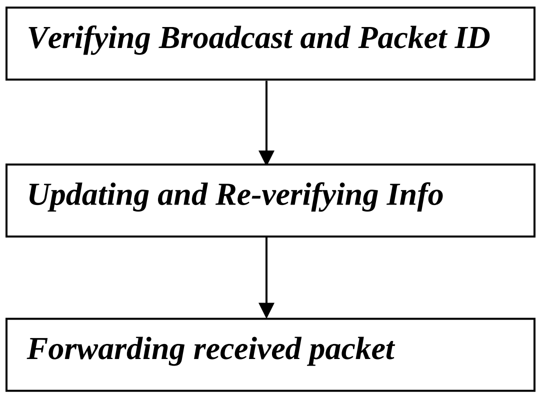

Verifying Broadcast and Packet ID
Updating and Re-verifying Info
Forwarding received packet

Figure 4.2.4. Node verifying broadcast and Packet ID on receiving packet.

If a route is found the following operations are performed in sequence before forwarding the data packet using the found route to the destination node as shown in figure 4.2.4A.

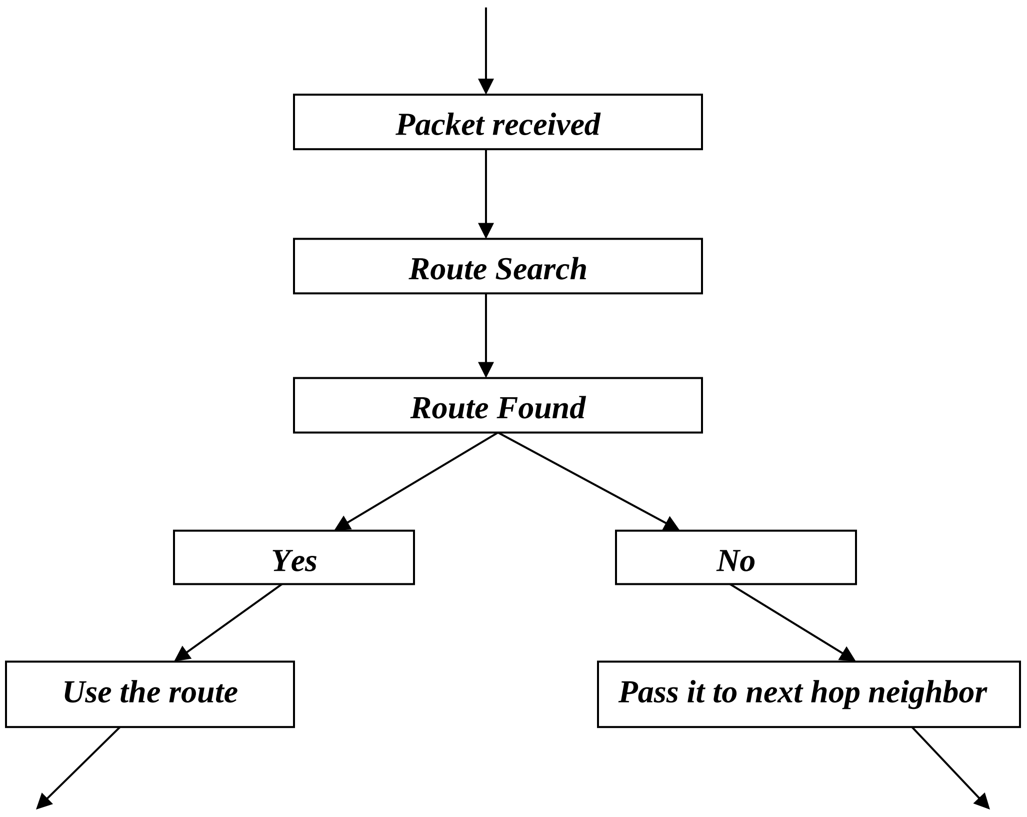


Figure 4.2.4A. Forwarding Packet Process of NFBTCP.

Verifying Broadcast and Packet ID. An intermediate node of acknowledged packet perform three operations in sequence, verifying broadcast and packet ID being the first function during which intermediate or acknowledge packet receiving node verify that it has not received the same packet before. If a packet with the same broadcast and packet ID has received before it is discarded and no further action is taken.

Updating and Re-verifying Information. Once it has confirmed that the packet with the same ID has not received before, receiver node updates relevant information about the sending node or other nodes of the network.

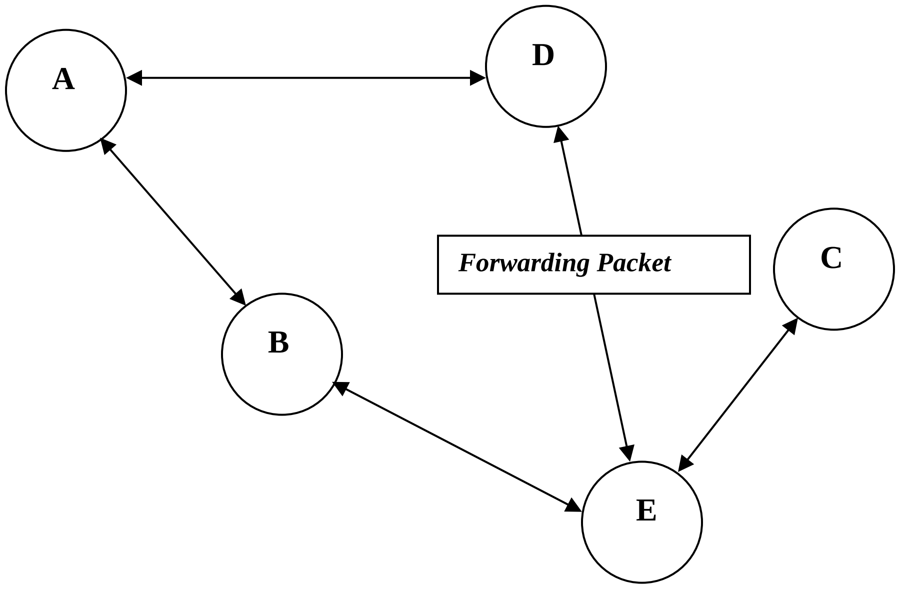


Figure 4.2.4B Node D forwarding packet to Node C

Forwarding Received Packet. If a route is found intermediate node forward the received packet using the available route as shown in figure 4.2.4B. In figure 4.2.4.B Node D forwarding received packet from Node A to Node C.

4.2.5. Link Failure Detection (LFD)

Mobile ad-hoc network by nature suffers with frequent topology changes and link failures happens unpredictably, detecting such failures in a mobile ad-hoc network is an important aspect to be seen. Such failure could be a means to degrading TCP performance over mobile ad-hoc network that makes this as an interrelated issue with the problem being investigated. It is assumed that the quicker we can detect such failures the better it could be for the network. Since, it could also add extra burden on the network via unnecessary data and control packets to the same route without knowing the route is broken. This could further leads to a point where network congestion could occur. If for a long time no packets are delivered and no acknowledgments are received, causing the TCP sender to reduce its window size dramatically, even though in fact no real congestion situation might exist.

NFBTCP introduces a new mechanism of updated notification to address this issue. The main aim of NFBTCP is the minimization of route failures, their prediction and a fast notification of the source in case of a route failure. In NFBTCP routing protocol which is used alongside TCP is made responsible for sending updated notifications whenever a link failure is detected. This could stop the sender sending any further packets using the broken route. A routing protocol is used alongside TCP which can also be made aware of the situation. In this case TCP will no longer be required to utilize its normal procedure of transporting packets via the same route. A specific action will be taken to communicate any updated link failure with TCP as effectively as possible in the implementation phase of this research project

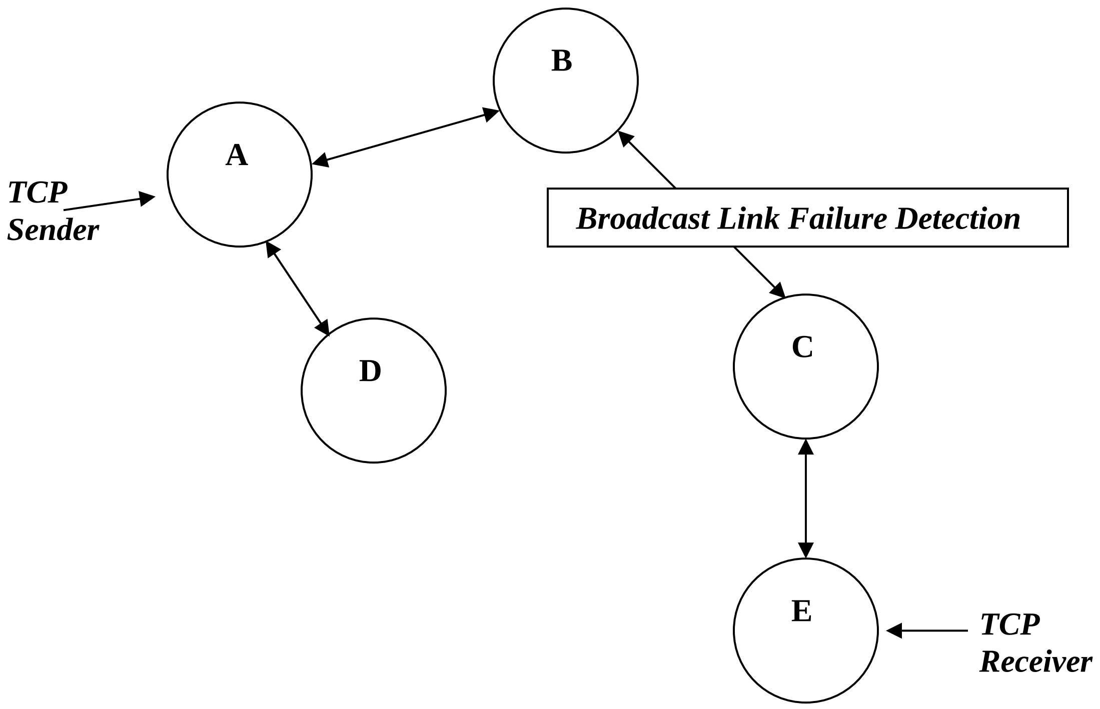


Figure.4.2.5 Link Failure Detection

```
Function Broadcast Link Failure

  Detect Link Failure
  Generate Update Packet
  Broadcast Update Packet

End Function
```

When a node detects a link failure, it broadcasts a link failure update packet across the network to inform other nodes about the link failure.

It could further be noted that Routing protocols for mobile ad-hoc network follows different strategies for route managements depend on the routing protocol used, this information can assist the routing protocol in the route management process and thus could make it easy to introduce such mechanisms in the overall communication structure of routing protocol used and TCP. It will be worthwhile to mention that AODV will be used alongside TCP to verify various concepts of the scheme alongside TCP in a simulation environment.

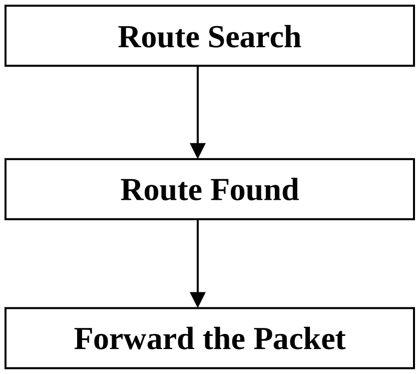


Figure 4.2.5a Route Search

Figure 4.2.5a shows the route search process where a search is made whenever a route request is received. If a suitable route is found the packet is forwarded to the destination.

4.2.5a Link Failure Update Packet (LFUP)

Link Failure Update Packet contains detail of failed route or a link. This packet is broadcast by an intermediate route which is in between the sender and the receiver of an active communication. A node on finding any route failure can broadcast LFUP. This could also update the routing tables of all other nodes in an active path.

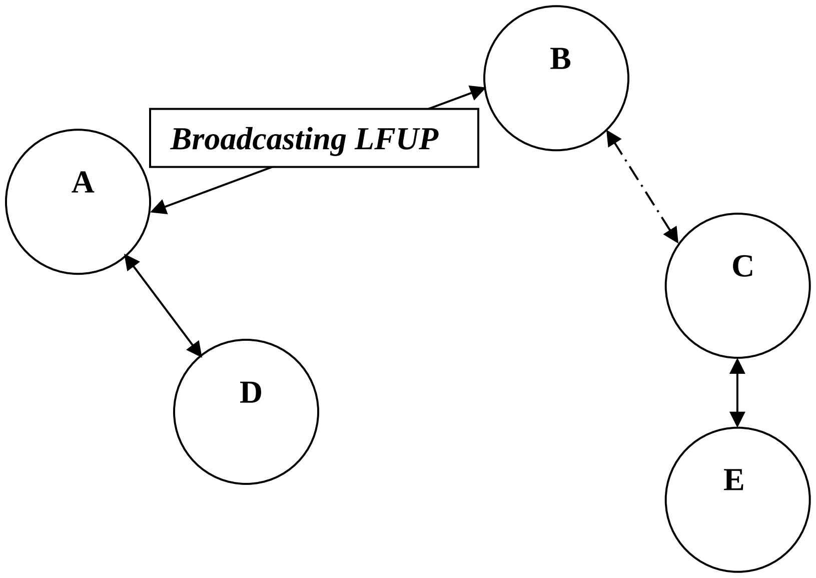


Figure 4.2.5b Node B is broadcasting LFUP

Link Failure Update Packet contains source sequence number, broadcast ID, destination sequence number and link breakage message as shown in figure 4.2.5c.

Source Seq no:	Broadcast ID	Destination Seq no:	Link Breakage Update message

Figure 4.2.5c LFUP Packet Structure

4.2.5b Link Update Packet (LUP)

Link update packet is broadcast when a new route is established between sender and destination node. Link update packet contains information about new link and is broadcast to all nodes involved in current communication. It could also be used to update routing table of all nodes in an active communication path. Upon receiving link update packet normal communication is resumed between source and destination nodes.

Source Seq no:	Broadcast ID	Destination Seq no:	New Link Update message

Figure 4.2.5d LUP structure

Link Update Packet contains source sequence number, broadcast ID, destination sequence number and new link update message as shown in figure 4.2.5d.

4.2.6. Link Capacity Detection (LCD)

It is well known in the context of mobile ad-hoc networks that link breakage happens frequently and unpredictably. This results in data loss and could also slow down the network speed. Protocols for mobile ad-hoc network deal with this problem in various manners. However in the case of TCP, TCP suffers from two main problems, congestion and slow start mechanism. Whenever TCP recovers from congestion or after retransmission timeout it invokes slow start mechanism. TCP shrinks its transmission rate to one segment (i.e. the size announced by the other end or the default, typically 512). Each time Acknowledgement is received, the congestion window is increased by one segment. The sender can transmit up to the maximum of the congestion window size.

In our scheme link capacity information is available at the routing table of the routing protocol. Additional parameters could be added in the existing specification of the routing protocol mentioned above to store such information. Therefore, whenever a new link is established nodes involved in active communication update their routing table with the link capacity information. In addition, all the nodes in between the sender and receiver are also updated.

TCP sender can get information from link formation packet stored in a routing table and can adjust its congestion window size accordingly. When a node detects new link it broadcast link formation packet (LFP), as shown in figure 4.2.6, containing link capacity information and is stored in a routing table of the routing protocol, therefore TCP doesn't need to invoke slow start mechanism when new link is detected and communication is resumed. Therefore, prior to a communication the TCP sender gets information about link capacity from the routing table of the next node involved in communication and adjusts its congestion window size.

It has been mentioned above that congestion could be avoided through the use of LFD operation of the proposed scheme. In this context LCD operation determines link capacity of a newly established link. In NFBTCP mobile nodes are made responsible to inform TCP sender about the new link capacity of the established link. This whole process is shown in figure 4.2.6 where a new link is established in between A and F via Node B and Node E, Please note that either Node E which is adjacent to the receiving node F will broadcast LFP.

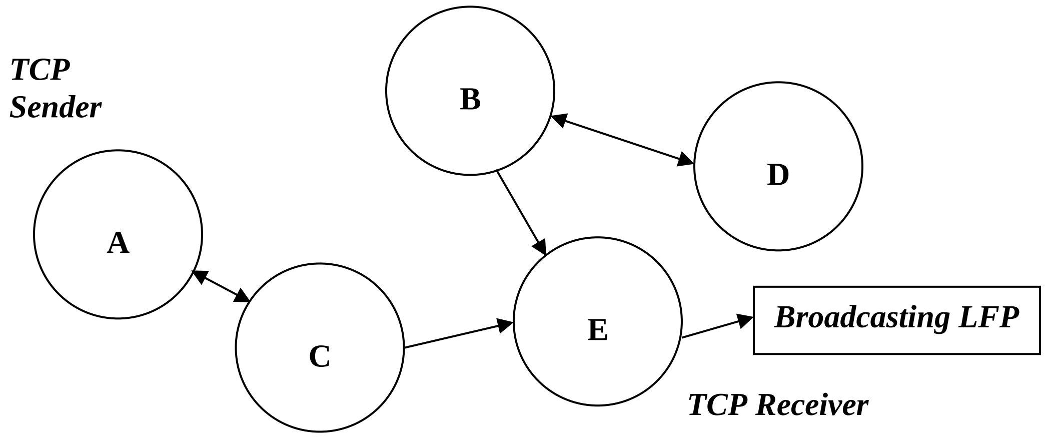


Figure 4.2.6 Broadcasting LFP

```
Function Link Capacity

  Calculate Link Capacity
  Update Link Capacity

End Function
```

Link capacity is calculated and is sent to the TCP sender so that adjustments could be made accordingly.

4.2.7 Link Formation Packet

Link formation packet contains information about the link capacity. This packet is broadcast in link capacity detection operation of the proposed scheme. In order to get link capacity information, link formation packet is broadcast and is stored in routing table of routing protocol. Routing protocols with cache can add an extra parameter to link formation packet and can store in its cache. Every time when a new link is detected link formation packet is updated with link capacity information. LFP contains source sequence number, broadcast ID, destination sequence number and link capacity information message as shown in figure 4.2.7.

Source Seq no:	Broadcast ID	Destination Seq no:	Link Capacity info: Message

Figure 4.2.7 LFP packet structure

4.2.8. out-of-order delivery Notification

It has been mentioned before that mobile ad-hoc networks suffer from frequent topology changes. TCP is known for in-order delivery to the receptionist; however no direct effective mechanism is known which can be used to deal with lost or dropped packets. This is of particular interest in the context of mobile ad-hoc networks, where packets could be dropped due to link or route failure. NFBTCP uses some of the known benefits to deliver solutions for the out-of-order delivery problem in mobile ad-hoc network environments.

In order to deal with out-of-order delivery of data packets a buffer is created in between TCP and the receiving node. Therefore rather than delivering a packet as it arrives all the packets of a single transmission are stored in the buffer. Likewise, TCP will be modified so that it can send one acknowledgement for the complete delivery of all the data packets of a single transmission rather than a single acknowledgement for a single packet. If a lost packet is detected a Buffer Update Packet will be sent to the sender pointing the missing packet as shown in figure 3.2.8 This lost packet can easily be identified either via sequence number or broadcast ID. Please note this information is normally included or assigned by the routing protocol of MANET. It is the responsibility of the sending node to re-broadcast the missing packet as identified by the TCP back to the receiver side using the same route as for the previous packet.

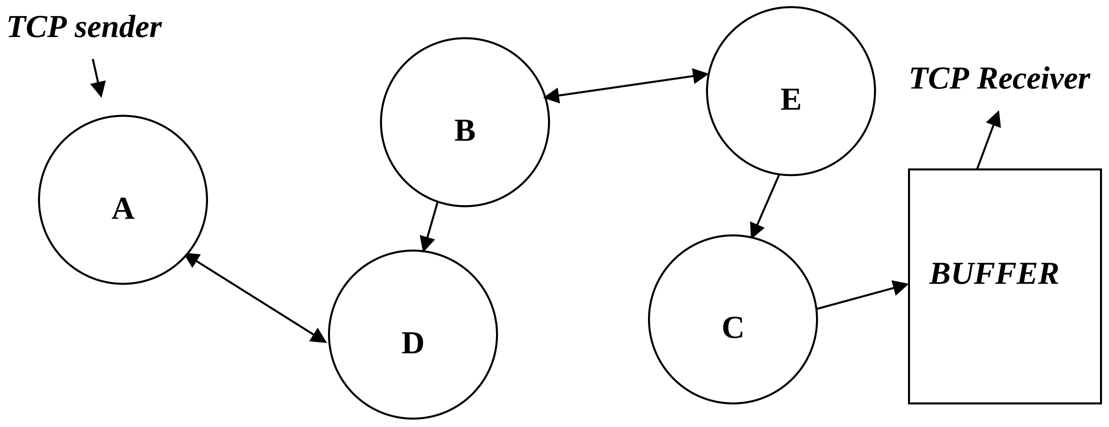


Figure.4.2.8 Broadcasting Buffer Update Packet

```
Function out-of-order delivery
    Discover missing packet sequence
    Requesting missing data packet
    Receive requested data packet
    Arranging data frame
    Delivering data to the destination

End function
```

When a packet sequence is found missing in an active data transmission a request to resend missing data packet is sent to the sender. On receiving the missing data packet the data frame is re organized and is delivered to the destination.

4.2.8a. Buffer Update Packet

Buffer Update Packet is sent whenever an out-of-order delivery is received at TCP buffer side. This serves an additional purpose and can also be used to update the intermediate nodes about the availability of the other nodes in between the source and the receiving node. Needless to mention such information is always fruitful and could also be used for any other possible communication by the intermediate nodes. Buffer update Packet contains source sequence number, broadcast ID, destination sequence number and out-of-order delivery notification message as shown in figure 4.2.8a.

Source Seq no:	Broadcast ID	Destination Seq no:	Out-of-order delivery Notification Message

Figure 4.2.8a BUP packet structure

4.3. Summary

In this chapter a detailed description outlining the main operations of NFBTCP is presented. The focus of this chapter is to establish a clear understanding related to the effectiveness of the proposed scheme. In this context, various functions as defined by the scheme specification are explained. In addition, all such explanations are aided by both the diagrammatic representation and pseudo codes of the individual functions. In view of the given scheme brief it can easily be understood that the scheme follows a unique and novel operational pattern in correspondence with the operational requirements of an ad-hoc network. It is quite clear that without implementation the scheme cannot be evaluated. Therefore, the proposed scheme is implemented in Java and evaluated in SWANS. The focus of the next chapter is on testing and evaluating NFBTCP in a simulation environment.

Chapter 5. Simulation and Implementation

5.1. Introduction

It was crucial to evaluate the implemented scheme to verify both concept and efficiency in varying as a simulation environment. ‘SWANS’ has been selected as simulation software. This decision is based on the fact that simulation of large number of Ad-hoc nodes is much easier to monitor in SWANS than in many other known simulators of similar type [41]. Simulation environments were created using different input parameters; it was due to aim of observing NFBTCP under different simulation environments. It is important to mention that the simulation covers most of the standard aspect of an evaluation cycle namely mobility, congestion control and physical network attributes. SWANS has certain limitation besides numerous benefits. Some of these weaknesses include its text based nature and usability. It is known that SWANS is easy to use with support of high scalability, however a user should have some previous understanding of the language in which software is written in order to have full benefits of the SWANS. In this chapter details of simulation experiments and a discussion of simulation results has been presented.

5.2. SWANS

Scalable wireless Ad-hoc network simulator (SWANS) runs over java in simulation time JiST platform. JiST is a high performance discrete event simulation engine that runs over a standard Java virtual machine [40]. The SWANS simulator combines the traditional systems-based and languages-based approaches to simulation construction. It was created primarily because existing network simulation tools are not sufficient for current research needs, and its performance serves as a validation of the virtual machine-based approach to simulator construction [41]. It is organized as independent software components that can be composed to form complete wireless network configurations.

5.2.1. Why SWANS was Chosen

Simulations play an important role in the development and evaluation of future systems. Researchers usually use simulations for the evaluation of ad hoc network protocols because they easily allow for a large number of nodes and reproducible environment conditions. Some of the existing well known network simulators are ns2, GloMoSim and SWANS. For the evaluation of NFBTCP, SWANS was chosen for simulation experiments. SWANS capabilities are similar to existing simulators but is able to simulate much larger networks and have a number of other advantages over existing tools. SWANS can run existing Java network applications, such as web servers and peer-to-peer applications, over the simulated network without modification. The application is automatically transformed to use simulated sockets and into a continuation-passing style [40]. The original network applications are run within the same process as SWANS, which increases scalability by eliminating the considerable overhead of process-based isolation. Network packets in SWANS are modeled as immutable objects, allowing a single copy to be shared across multiple nodes. This saves the memory and time of multiple packet copies on every transmission. A diagram illustrating how SWANS works is shown in figure 5.In SWANS, simulation events among the various entities such as packet transmissions are performed with no memory copy and no context switch. The system also continuously profiles running simulations and dynamically performs code in lining, constant propagation and other important optimizations, even across entity boundaries. This is important, because many stable simulation parameters are not known until the simulation is running. Memory is critical for simulation scalability. Automatic garbage collection of events and entity state in SWANS not only improves robustness of long-running simulations by preventing memory leaks, it also saves memory by facilitating more sophisticated memory protocols. An example of memory savings in SWANS is the use of soft references for storing cached computations, such as routing tables. These routing tables can be automatically collected, as necessary, to free up memory. SWANS is unique in a way that it allows integration of new scheme within the existing model more comfortably than some other simulators such as ns2. In addition, SWANS is capable of running simulations involving thousands of mobile nodes with results that match the accuracy of other well known simulation software. In addition, almost all of the mobility models are available in the software in order to design simulation configuration of one choice. In the light of the above it is clear that SWANS closely matches the requirements of this research project. Thus it is selected as a simulation tool for evaluation of NFBTCP. The coded file of the proposed implemented scheme is included in SWANS. A script file is written specifying various parameters as needed in a simulation experiments. The software is then recompiled in order to take effects of the changes. The same processes were followed in the case of NFBTCP.

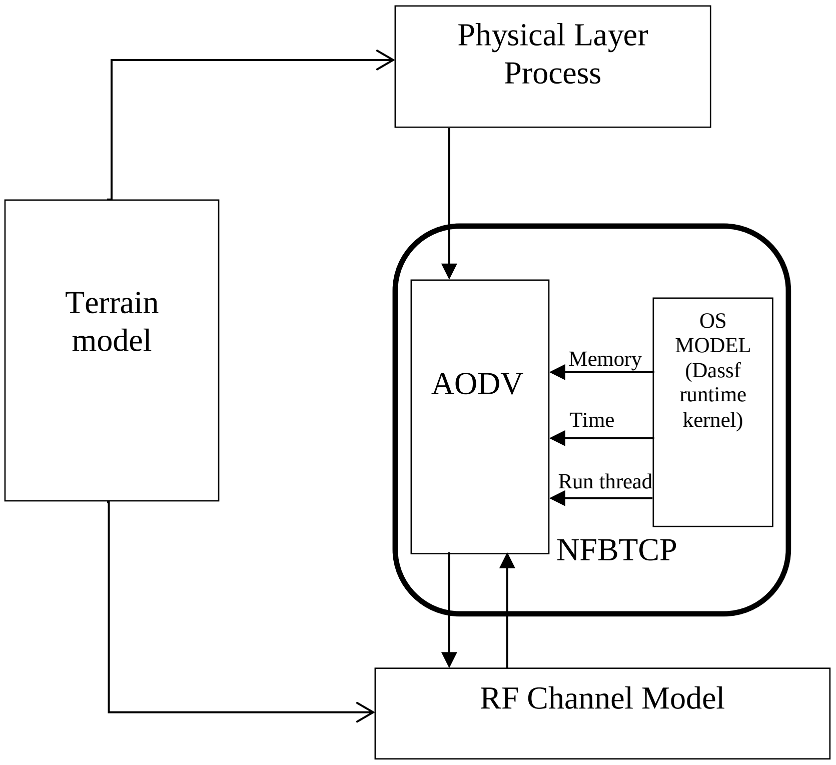


Figure 5. SWANS structure.

5.3. Implementation

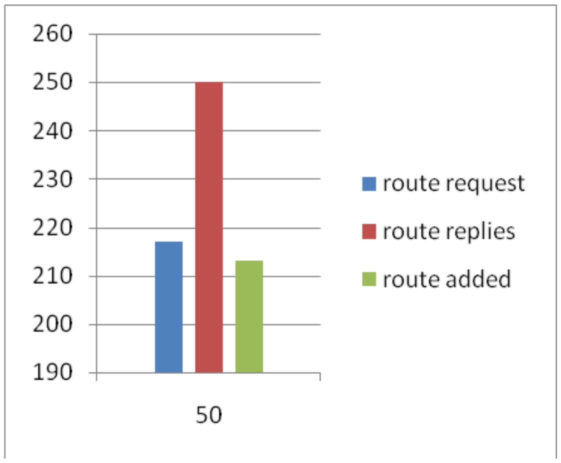
Implementation of defined functions has been done in Java using a single file NFBTCP.Java. This single file is later added in SWANS which is then re-complied and simulation experiments were run using AODV as a routing protocol. Details of simulation and results are also explained in this chapter.

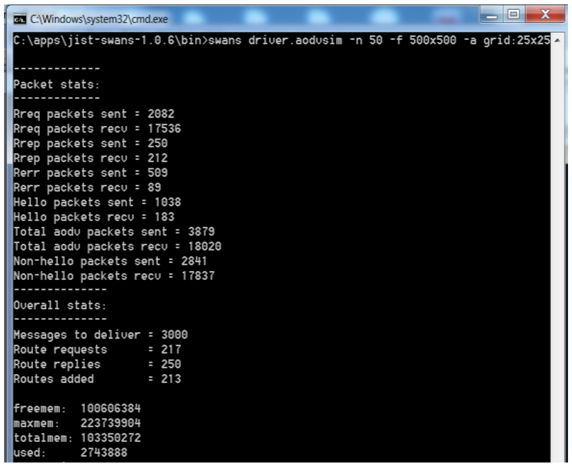
5.3.1. Implementation Language

Java language has been selected as an implementation language to program the proposed scheme. It is due to the fact that the proposed scheme will be evaluated in SWAN simulator software which is written in Java. Java is an object oriented language which supports inheritance, polymorphism and message passing etc. at present it is famous in mobile phones and bank applications. Java also offers independent architecture and many enterprise applications are being developed in Java. A class implementing interface is defined prior to a static class referring to the actual interface class of the simulation software. This has been followed by a constructor where variables as defined in the actual code files are initialized. Functions have been written which reflect specification and network operational patterns which can be seen in the appendix. It is important to mention that Java has been selected as the implementation language due to simulation software programming which is done in Java.

5.4. Simulation Experiments

Evaluation experiments were conducted on Windows under SWANS and various input parameters were used to monitor the developed scheme. A single simulation script with the reference to NFBTCP implementation file is created and for each simulation experiment different nodes and mobility patterns were used. In total eight different experiments were conducted. In each of these experiments varying mobile nodes, field and grid size were used. It is important to note that packet loss was defined as none. Since declaring packet lost means nothing to the simulator. From the first until the fourth experiment numbers of nodes were increased to 200 with a fixed field size of 500 and grid size of 25. In the second set of experiments nodes were increased in the same manner as was in the first set of experiments. However, the field size was decreased to 250 and grid size was decreased to 15. In the light of evaluation experiments it can be seen that all functions as defined by NFBTCP is fully applicable and operational. In the first set of experiments a higher number of route requests and route replies were observed as can be seen in figure5.1. In addition 213 routes were added as can be seen in figure 5.1

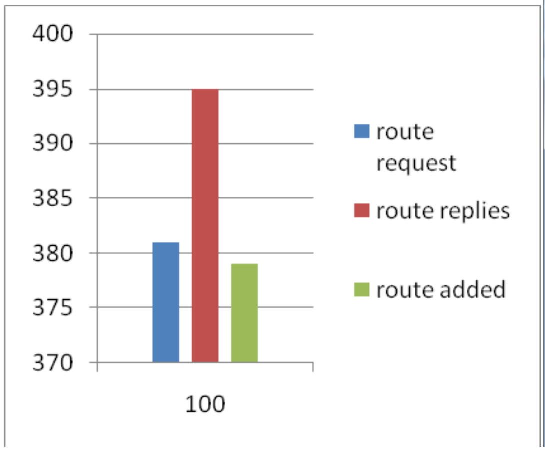


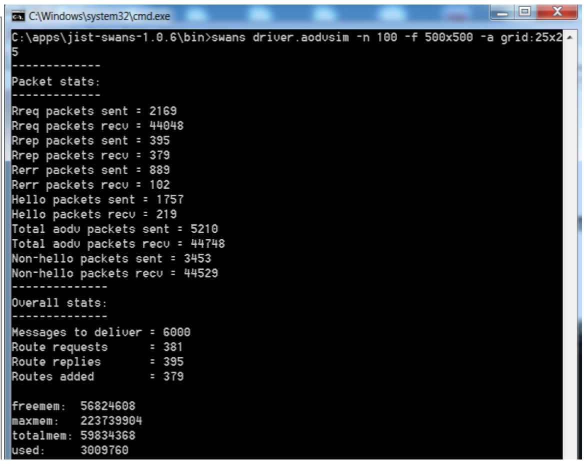


```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 50 -f 500x500 -a grid:25x25

-------------
Packet stats:
-------------
Rreq packets sent = 2082
Rreq packets recv = 17536
Rrep packets sent = 250
Rrep packets recv = 212
Rerr packets sent = 509
Rerr packets recv = 89
Hello packets sent = 1038
Hello packets recv = 183
Total aodv packets sent = 3879
Total aodv packets recv = 18020
Non-hello packets sent = 2841
Non-hello packets recv = 17837
--------------
Overall stats:
--------------
Messages to deliver = 3000
Route requests      = 217
Route replies       = 250
Routes added        = 213

freemem:  100606384
maxmem:   223739904
totalmem: 103350272
used:     2743888
```

Figure 5.1 simulatin with 50 mobile nodes ***screenshot of experiment 1***

In the second set of experiment nodes were increased to 100 with a fixed field and grid size. An increase in the route requests and route replies were observed in addition to the higher number of routes which were added at the end of simulation cycle. These could also be observed in figure 5.2.

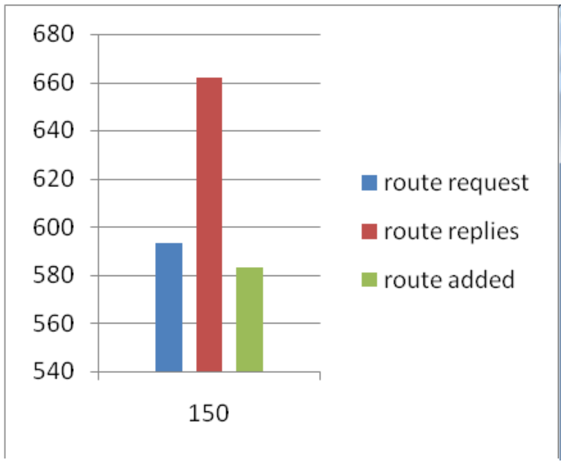


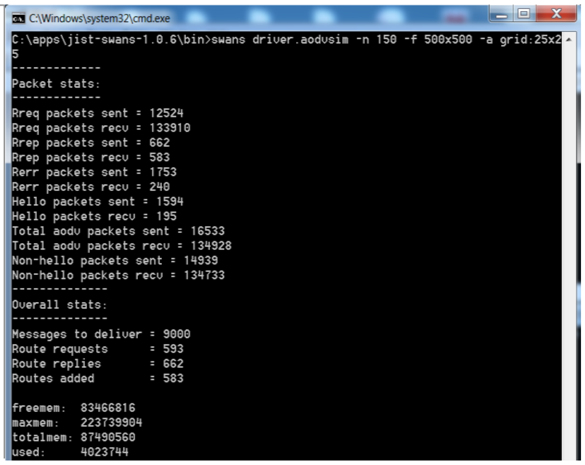


```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 100 -f 500x500 -a grid:25x2
5
-------------
Packet stats:
-------------
Rreq packets sent = 2169
Rreq packets recv = 44048
Rrep packets sent = 395
Rrep packets recv = 379
Rerr packets sent = 889
Rerr packets recv = 102
Hello packets sent = 1757
Hello packets recv = 219
Total aodv packets sent = 5210
Total aodv packets recv = 44748
Non-hello packets sent = 3453
Non-hello packets recv = 44529
--------------
Overall stats:
--------------
Messages to deliver = 6000
Route requests      = 381
Route replies       = 395
Routes added        = 379

freemem:  56824608
maxmem:   223739904
totalmem: 59834368
used:     3009760
```

Figure 5.2 simulatin with 100 mobile nodes ***screenshot of experiment 2***

In the third set of experiment higher message activity in terms of route requests and route replies were observed in comparison with the above two experiments as it was expected due to an increased in number of nodes. Likewise, higher numbers of routes were added as shown in figure 5.3. This show a very good behavior as it was expected.

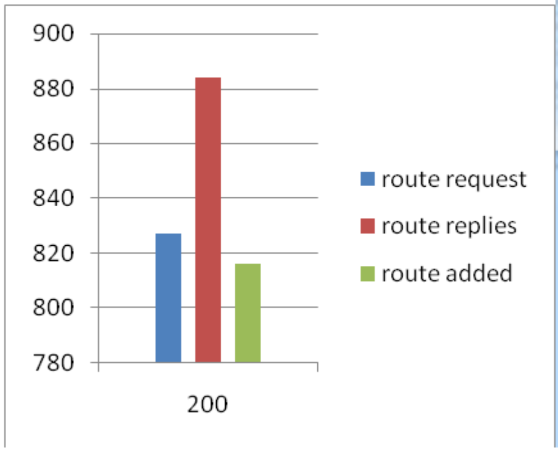


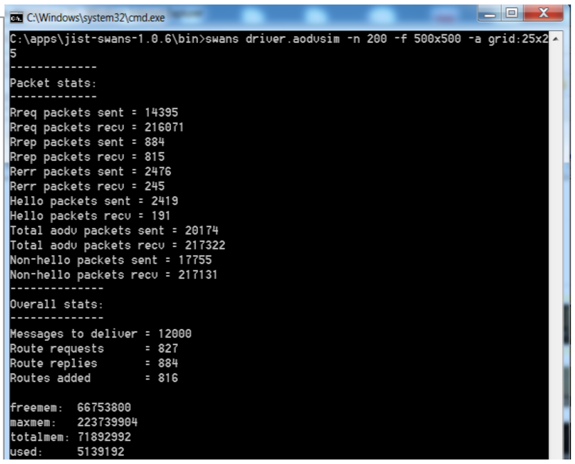


```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 150 -f 500x500 -a grid:25x2
5
-------------
Packet stats:
-------------
Rreq packets sent = 12524
Rreq packets recv = 133910
Rrep packets sent = 662
Rrep packets recv = 583
Rerr packets sent = 1753
Rerr packets recv = 240
Hello packets sent = 1594
Hello packets recv = 195
Total aodv packets sent = 16533
Total aodv packets recv = 134928
Non-hello packets sent = 14939
Non-hello packets recv = 134733
--------------
Overall stats:
--------------
Messages to deliver = 9000
Route requests      = 593
Route replies       = 662
Routes added        = 583

freemem:  83466816
maxmem:   223739904
totalmem: 87490560
used:     4023744
```

Figure 5.3 simulatin with 150 mobile nodes ***screenshot of experiment 3***

In the fourth set of experiment nodes were increased to 200 with a fixed field size of 500 and grid size of 25. In these set of experiments the higher message activity in terms of route requests and route replies were observed due to an increase in number of nodes as shown in figure 5.4. Likewise, higher numbers of routes were added.

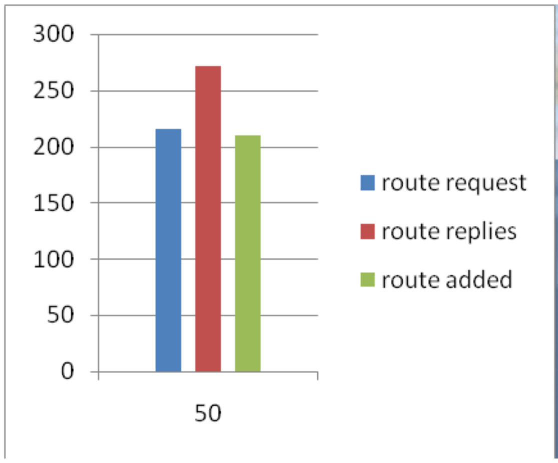


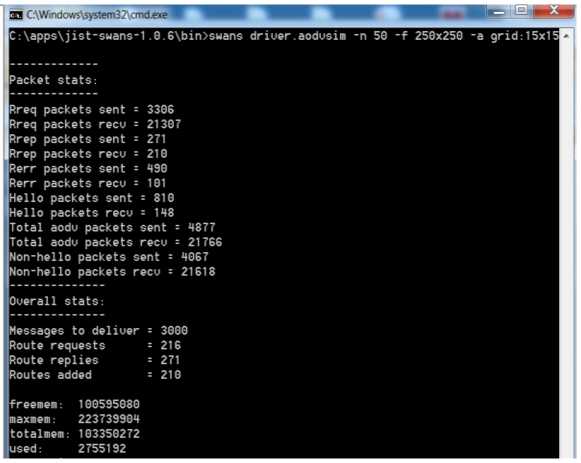


```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 200 -f 500x500 -a grid:25x2
5
-------------
Packet stats:
-------------
Rreq packets sent = 14395
Rreq packets recv = 216071
Rrep packets sent = 884
Rrep packets recv = 815
Rerr packets sent = 2476
Rerr packets recv = 245
Hello packets sent = 2419
Hello packets recv = 191
Total aodv packets sent = 20174
Total aodv packets recv = 217322
Non-hello packets sent = 17755
Non-hello packets recv = 217131
--------------
Overall stats:
--------------
Messages to deliver = 12000
Route requests      = 827
Route replies       = 884
Routes added        = 816

freemem:  66753800
maxmem:   223739904
totalmem: 71892992
used:     5139192
```

Figure 5.4 simulatin with 200 mobile nodes ***screenshot of experiment 4***

In the remaining four sets of experiments mobile nodes were increased from 50 to 200 however the fixed field size was decreased to 250 and grid size was decreased to 15. In the fifth set of experiment result shows same high performance as with the double field and grid sizes as shown in figure 5.5.

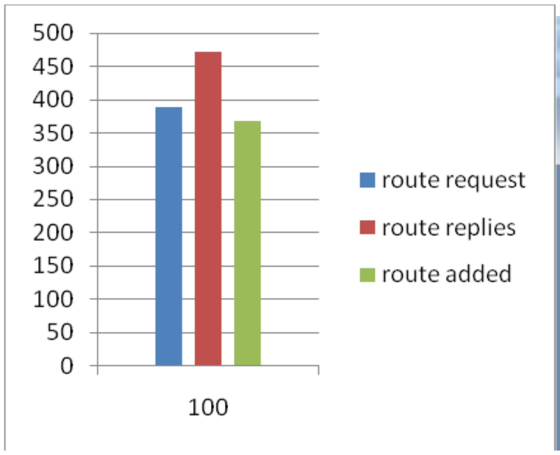


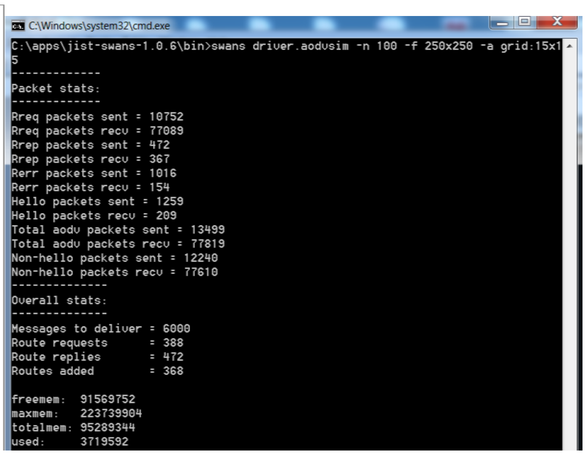


```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 50 -f 250x250 -a grid:15x15

-------------
Packet stats:
-------------
Rreq packets sent = 3306
Rreq packets recv = 21307
Rrep packets sent = 271
Rrep packets recv = 210
Rerr packets sent = 490
Rerr packets recv = 101
Hello packets sent = 810
Hello packets recv = 148
Total aodv packets sent = 4877
Total aodv packets recv = 21766
Non-hello packets sent = 4067
Non-hello packets recv = 21618
--------------
Overall stats:
--------------
Messages to deliver = 3000
Route requests      = 216
Route replies       = 271
Routes added        = 210

freemem:  100595080
maxmem:   223739904
totalmem: 103350272
used:     2755192
```

Figure 5.5 simulatin with 50 mobile nodes ***screenshot of experiment 5***

In the sixth set of experiment nodes were increased to 100 with a fixed field and grid size. An increase in the route requests and route replies were observed in addition to the higher number of routes which were added at the end of simulation experiments as shown in figure 5.6.

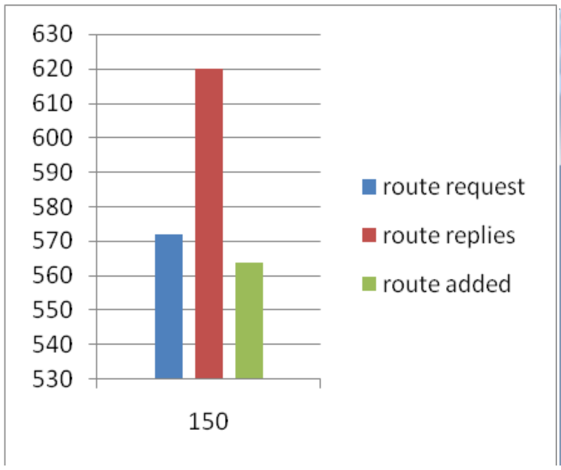


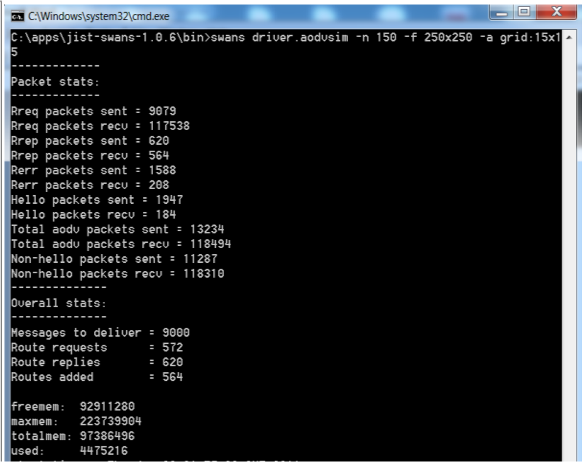


```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 100 -f 250x250 -a grid:15x1
5
-------------
Packet stats:
-------------
Rreq packets sent = 10752
Rreq packets recv = 77089
Rrep packets sent = 472
Rrep packets recv = 367
Rerr packets sent = 1016
Rerr packets recv = 154
Hello packets sent = 1259
Hello packets recv = 209
Total aodv packets sent = 13499
Total aodv packets recv = 77819
Non-hello packets sent = 12240
Non-hello packets recv = 77610
--------------
Overall stats:
--------------
Messages to deliver = 6000
Route requests      = 388
Route replies       = 472
Routes added        = 368

freemem:  91569752
maxmem:   223739904
totalmem: 95289344
used:     3719592
```

Figure 5.6 simulatin with 100 mobile nodes ***screenshot of experiment 6***

In the seventh set of experiment nodes were increased to 150 with a fixed field and grid size. Higher message activity in terms of route requests and route replies were observed in comparison with the above two experiments with the same fixed field and grid size, as it was expected due to an increased in number of nodes. Likewise, higher numbers of routes were added as shown in figure 5.7.

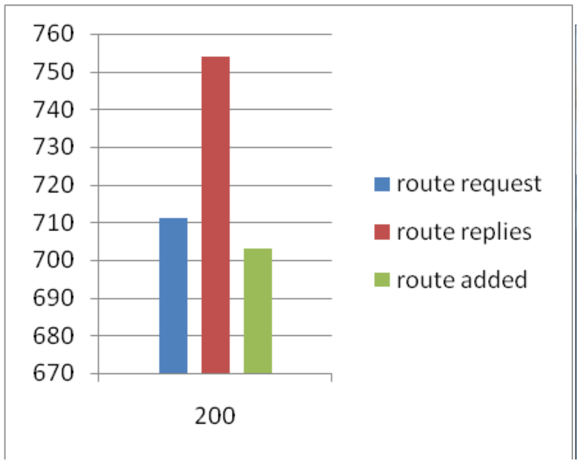


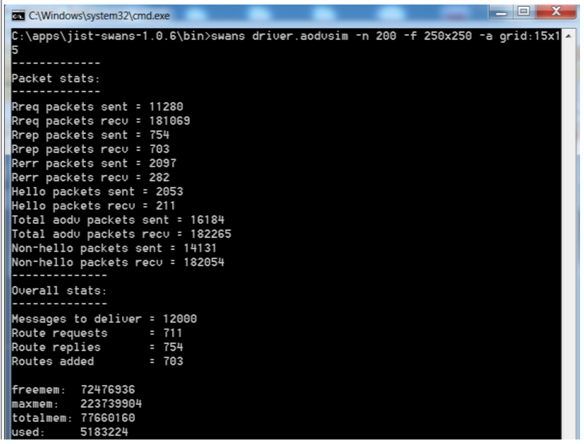


```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 150 -f 250x250 -a grid:15x1
5
-------------
Packet stats:
-------------
Rreq packets sent = 9079
Rreq packets recv = 117538
Rrep packets sent = 620
Rrep packets recv = 564
Rerr packets sent = 1588
Rerr packets recv = 208
Hello packets sent = 1947
Hello packets recv = 184
Total aodv packets sent = 13234
Total aodv packets recv = 118494
Non-hello packets sent = 11287
Non-hello packets recv = 118310
--------------
Overall stats:
--------------
Messages to deliver = 9000
Route requests      = 572
Route replies       = 620
Routes added        = 564

freemem:  92911280
maxmem:   223739904
totalmem: 97386496
used:     4475216
```

Figure 5.7 simulatin with 150 mobile nodes ***screenshot of experiment 7***

In the eight set of experiment nodes were increased to 200 with a fixed field and grid size. Higher route requests and route replies were observed due to an increased in number of nodes. Likewise, higher numbers of routes were added as shown in figure 5.8.





```
C:\apps\jist-swans-1.0.6\bin>swans driver.aodvsim -n 200 -f 250x250 -a grid:15x1
5
-------------
Packet stats:
-------------
Rreq packets sent = 11280
Rreq packets recv = 181069
Rrep packets sent = 754
Rrep packets recv = 703
Rerr packets sent = 2097
Rerr packets recv = 282
Hello packets sent = 2053
Hello packets recv = 211
Total aodv packets sent = 16184
Total aodv packets recv = 182265
Non-hello packets sent = 14131
Non-hello packets recv = 182054
--------------
Overall stats:
--------------
Messages to deliver = 12000
Route requests      = 711
Route replies       = 754
Routes added        = 703

freemem:  72476936
maxmem:   223739904
totalmem: 77660160
used:     5183224
```

Figure 5.8 simulatin with 200 mobile nodes ***screenshot of experiment 8***

5.5. Comparison and Discussion

As mentioned previously a systematic approach has been followed from the start to the end of the project. A special focus was given on the meeting the defined aims and objectives of this project. These aims and objectives are transformed into milestones and each of these milestone are evaluated as a sign of marking project progress. In addition, all of these objectives are achieved as a project outcome. In the context of this particular project, evaluation observation details one of the two things related to the proposed and implemented scheme. The first in between these two factors is the confirmation of the theoretical concepts which were included as a part of functional specification of NFBTCP. It can be confirmed that NFBTCP has proven itself as a fully functional and operational-able for mobile ad-hoc network, thus should be seen or taken as a new novel TCP based solution for mobile ad-hoc network. The second and the last aspect is the scheme performance. In the light of the above given experiments details, it could be seen that NFBTCP performed well in all simulation environment.

It could be noted from the light of the literature review that there are a number of different schemes such as TCP-F, TCP-ELFN, ATCP and Split-TCP etc, were proposed. However, it is known that these schemes in one way or the other lack a number of known operational requirements and thus do not fully address TCP performance within mobile ad-hoc networks. In TCP-F the RRN packet is generated when the intermediate node detects re-establishment of broken path and it depends on information from routing protocol. TCP-F has an additional state compared to the traditional TCP state machine, and hence its implementation requires modifications to the existing TCP libraries. Another disadvantage of TCP-F is that the congestion window used after a new route is obtained may not reflect the achievable transmission rate acceptable to the network and the TCP-F. In TCP-ELFN when the network is temporarily partitioned, the path failure may last longer; this can lead to the origination of periodic probe packets consuming bandwidth and power. Another disadvantage is that the congestion window used after a new route is obtained may not reflect the achievable transmission rate acceptable to the network and the TCP receiver. ATCP depend on the network layer protocol to detect the route changes and partitions, which not all routing protocols may implement. Addition of a thin ATCP layer to the TCP/IP protocol stack requires changes in the interface functions. Split-TCP requires modifications to TCP protocol. The end-to-end connection handling of traditional TCP is violated. The failure of proxy nodes or frequent path breaks, affects the performance of split-TCP. In view of simulation results where NFBTCP showed better throughput performance as more routes available in an ad hoc network implies possibility of higher throughput since more data can be transferred using the established routes. It does however require a more practical demonstration of comparison with other schemes which is included as a part of potential future PhD study.

A higher number of route requests and route replies representing networking activities were observed with the increase of mobile nodes. This clearly showed that NFBTCP fits well within mobile ad-hoc networking environments. Since increase of mobile nodes to some extent implies increase in the communication taking place in a network. In addition to the messages activities, good numbers of routes were added at the end of each simulation cycle. It is quite understandable that the more routes available for data transfer in a mobile ad-hoc network, the better. Moreover, such additions to the available routes could directly impact overall throughput. It is due to the nature of mobile ad-hoc networks, where routes forms and are broken almost unexpectedly. Therefore an alternative route to the destination is always beneficial. Lastly, nodes in mobile ad-hoc networks suffer with limited resources. That makes conservation of all such resources an important issue in the context of mobile ad-hoc networks.

This research analyzed TCP performance over mobile ad-hoc network prior to proposing a functional solution to address identified weaknesses. In this context, both TCP and previously reported schemes have been taken into consideration so as to build a fully effective model. Thus NFBTCP has been a more reliable and effective TCP based solution for mobile ad-hoc networks. In NFBTCP, TCP specific functions are modified in NFBTCP specification to solve TCP issues in mobile ad-hoc network. Examples of such main functions are Generating Acknowledgement, Generating Broadcast and Packet ID, Link Failure Detection, Link Capacity Detection and out-of-Order Delivery Notification.

Generating acknowledgement process in NFBTCP is achieved via acknowledge packet (AP) delivery from destination to the source node of a packet. When an intermediate node receives a packet from a source node for destination node verify broadcast and packet ID that it has not received the same packet before it is discarded. If the same packet is not received the intermediate node stores it and sends acknowledge pack to the sender and forwards the packet received from the source node to the destination node. Thus, nodes in NFBTCP maintain fresh information about other nodes and can be used to send control acknowledgements. Therefore, TCP does not need to send acknowledgements as normal; rather it is replaced by the control acknowledgement of NFBTCP. Therefore, TCP does not need to shrink its congestion window due to TCP sender not receiving acknowledgement rather it is directly handled by NFBTCP.

In NFBTCP when a destination node or intermediate node responsible for forwarding packet to the destination node detects link breakage it inform the TCP sender about link breakage so that the TCP sender stops sending further packets to the destination unless a new route is established. Detecting link failures is very important and informing the TCP sender about the link failure can improve network throughput. In NFBTCP intermediate nodes between the sender and destination nodes are made responsible for responding quickly to link failures and broadcast link failure update packets containing link failure information. All nodes involved in current communications update their routing tables upon reception of an LFUP. In NFBTCP when a new link is established a link update packet is broadcast to inform the source node about the new established route to allow the sending packets to resume

NFBTCP introduces a link capacity detection function to detect the link capacity of communication between source and destination nodes. A link formation packet is used to inform the TCP sender about link capacity so that it adjusts its congestion window according to information available in the link formation packet, thus improving overall throughput of the network. NFBTCP uses some of the known benefits to deliver solutions to the out-of-order delivery problem of TCP in mobile ad-hoc network environments. All the packets of a single transmission are stored in a buffer created between TCP and the receiving node and a send single acknowledgement is sent to the TCP sender. The buffer update packet contains information about lost packets and will be sent to the TCP sender pointing out missing packets and this lost packet will be identified by either sequence number or broadcast ID. Thus TCP sender obtains information from buffer update packets to resend it to the sender. In summary, in the light of the NFBTCP specification and above explanation it is quite clear that NFBTCP offers an excellent TCP based solution for communication over mobile ad-hoc network.

In the following chapter conclusions which are drawn from the start until the end of this research project and directions for future work are presented.

5.6. Summary

In this chapter a detailed account of the evaluation phase of the proposed and developed scheme is presented. The evaluation results clearly demonstrate the effectiveness of NFBTCP in addressing issues of TCP performance in mobile ad-hoc networks. In view of the obtained results it can easily be observed that some of the known TCP issues in mobile ad-hoc networks are effectively addressed by NFBTCP. Moreover, a good throughput at the end of various simulation cycles indicates a notable performance for NFBTCP over mobile ad-hoc networks. The focus of the next chapter is on concluding research findings which are gathered at various stages of this project.

Chapter 6. Conclusions and Future Work

6.1. Introduction

This work could be seen as a series of associated tasks which were carried out before being reached to the final solution of the problem being investigated. In this context phase one could include researching the main problem domain and analyzing the existing solutions. After analyzing the existing solutions certain shortcomings were identified in this area and have been selected to work on as a part of this project. The phase two comprises proposing a novel solution to the problem, design, implementation and evaluation. NFTCP follows an intermediate approach in between some of the existing mechanisms of TCP based schemes, different functions were developed to improve TCP performance over mobile ad-hoc network, thus should be taken as novel based solution for mobile ad-hoc network. The novelty of NFBTCP is the most important aspect of this project. The network formation of NFBTCP is an additional function added to the TCP structure for the formation of the mobile ad-hoc network. NFBTCP also deals with congestion avoidance in a TCP based transmission. The effect of the link breakage is one of the constraints of TCP over mobile ad-hoc network. Detecting link breakage as soon as it happens and informing the sender about the link breakage improves TCP performance. In NFBTCP nodes destination node or nodes part of communication inform sender about broken path to stop sending further packets in order to avoid congestion and packet lost due to retransmission timeout. In NFBTCP a buffer is created at destination node which stores all packets and if a packet is lost or out of order packets arrived at destination node request the missing packet and single acknowledgement is sent to the sender after arrival of the packets in a data frame. Implementation of the scheme has been done in java and evaluated using SWANS. Simulation results show more routes were added at the end of each experiment. Alternative route to the destination is always valuable in mobile ad-hoc networks where routes are form and broken frequently. The simulation experiment results demonstrate that the main concepts have been successfully met as underlined in the proposed scheme, especially congestion control and out-of-order packet delivery. The higher number of routes with the increase of mobile nodes suggests that the implemented congestion avoidance mechanism is functional with a reduction in link breakage since otherwise it would not be possible for more routes to be added. The congestion control process affects the smooth flow of data and control packets. Similarly, the addition of mobile nodes illustrates that more packets are broadcasted. A route can be established each time a request for route formation is sent. Therefore continuation of route formation explains why the flow was normal in a simulation cycle. Thus the proposed scheme addresses congestion avoidance as expected. In order to avoid out-of-order packet delivery a buffer was introduced into NFBTCP; more throughput at the end of each simulation cycle also suggests that the idea of adding state worked well in the scenarios considered. Conclusions can be seen as a combination of these two phases and are presented in section 2 where directions of future work are given in section 3 of this chapter.

6.2. Conclusions

Routing in mobile ad-hoc networks is a challenging issue. Much effort is going on to invent routing mechanisms which can fulfill the typical routing requirements of mobile ad-hoc networks. TCP/IP has initially been designed to support similar tasks in fixed networks. Work has been done to transform TCP as a routing solution for mobile ad-hoc networks. This research has discovered that most of the TCP based variant routing solutions of mobile ad-hoc networks have not been successful in addressing the problem fully. Taking TCP based routing solutions as a main problem; this research has proposed a novel routing solution called Node feedback TCP based as a routing scheme for mobile ad-hoc networks. The scheme has been developed in Java and evaluated using SWANS. The following are the conclusions driven in the light of the conducted research.

Evaluation observations detail the confirmation of the theoretical concepts which were included as a part of the functional specifications of NFBTCP. NFBTCP has validated itself as a fully functional and operational-able for mobile ad-hoc networks, thus should be taken as a new novel TCP based solution for mobile ad-hoc networks. In the light of the conducted experiments, it can be seen that NFBTCP performed well in all simulation environments. A higher number of route requests and route replies representing networking activities were observed with the increase of mobile nodes. This clearly showed that NFBTCP fits well within a mobile ad-hoc networking environment. Increase of mobile nodes to some extent implies increase in the communication taking place in a network. In addition to the messages activities, a good numbers of routes were added at the end of each simulation cycle. It is quite understandable that the more routes available for data transfer in a mobile ad-hoc network, the better. In this context unlike traditional TCP better route connectivity is possible through NFBTCP.

It is important to mention that more routes have a direct connection with the overall throughput. In mobile ad-hoc networks routes form and are broken almost unexpectedly, hence an alternative route to the destination is always beneficial. Nodes in mobile ad-hoc networks suffer from limited resources it makes conservation of all such resources an important issue in the context of mobile ad-hoc networks.

In relation to the literature review TCP invokes congestion control mechanism unnecessarily. This has been addressed through a newly introduced congestion control mechanism of NFBTCP. In NFBTCP it is the responsibility of the node forwarding packet to the destination node upon detecting route breakage to inform the TCP sender about route breakage so that the TCP sender does not invoke the congestion control mechanism. The TCP sender should wait until a new route is established instead of invoking the congestion control mechanism. This approach is more feasible and close to the ad hoc network operational requirement. Link failure due to frequent topology changes is the major factor affecting TCP performance over mobile ad-hoc. This could further lead to a point where network congestion could occur. In NFBTCP routing protocol which is used alongside TCP is made responsible for sending updated notifications whenever a link failure is detected. A link Failure Update Packet is broadcast to inform the TCP sender about link failures. This could also update the routing tables of all other nodes on an active path.

6.3. Future Work

NFBTCP has been designed and implemented in a manner which allows integration of other schemes with the implemented. In addition, it has proven itself as a suitable and efficient scheme in view of the implementation and simulation phase. However, the following are some of the directions which could be taken in any possible modification of the developed scheme.

- **Extending NFBTCP Support.** NFBTCP can be overlooked in order to extend it support for sister networks of an ad-hoc network. Since, most of the transportation concepts are similar at a normal level.
- **Evaluation of NFBTCP with Routing Schemes.** In the presented research, NFBTCP has been simulated with AODV, it has always been a good area to further monitor scheme performance with some other schemes such as DSR, MAODDP and DSDV.
- **Security.** This has been a challenging issue in the context of mobile ad-hoc networking. Though many secure routing schemes [MAODDP] have been proposed however, it would be good to look into it from a different perspective by addressing security at a transport level.
- **Hidden Terminal Problem.** NFBTCP has addressed several TCP related issue without being too deep into MANET other related problem. It does make sense because such topics fall outside the scope of the conducted research. However, it could provide a complete new vision to the whole scenario if this particular problem is addressed in view of further enhancing NFBTCP performance of handing such issues.

References

[1] M. Issoufou Tiado, R. Dhaou and A.-L. Beylot ‘’RCL : A new Method for Cross–Layer Network Modelling and Simulation’’ ENSEEIHT – IRIT Lab. 2 rue C. Camichel, 2005

[2] Raisinghani, V.T.; Iyer, S ‘’Cross-layer feedback architecture for mobile device protocol stacks’’ Communications Magazine, IEEE Volume 44, Issue 1, Page: 85 – 92. 2006

[3] Jaehoon Kim and Kwangsue Chung ‘’C-Snoop: Cross Layer Approach to Improving TCP Performance Over Wired and Wireless Networks’’ School of Electronics Engineering, 2007

[4] S. Shakkottai, T. S. Rappaport, and P. C. Karlsson, "Cross-layer design for wireless networks," IEEE Communications Magazine, vol. 41, pp. 74-80, 2003.

[5] V. T. Raisinghani and S. Iyer, "Cross-layer design optimization in wireless protocol stacks," Computer Communications, vol. 27, pp. 720-724, 2004.

[6] L. Chen, S. H. Low, and J. C. Doyle, "Joint congestion control and media access control design for ad hoc wireless networks," presented at IEEE INFOCOM, 2005.

[7] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar, "ATP: A reliable transport protocol for ad-hoc networks," presented at MOBIHOC: PROCEEDINGS OF The Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing, 2003.

[8] S. Kopparty, S. V. Krishnamurthy, M. Faloutsos, and S. K. Tripathi, "Split TCP for mobile ad hoc networks," presented at GLOBECOM'02 - IEEE Global Telecommunications Conference, 2002.

[9] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "Feedback based scheme for improving TCP performance in ad-hoc wireless networks," presented at Proceedings of the 18th International Conference on Distributed Computing Systems, 1998.

[10] X. Yu, "Improving TCP performance over mobile ad hoc networks by exploiting cross-layer information awareness," presented at MobiCom Proceedings of the Tenth Annual International Conference on Mobile Computing and Networking, 2004.

[11] Dan Liu, Mark Allman, Shudong Jin, Limin Wang, “Congestion Control Without a Startup Phase”, Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), 2007

[12] J. Liu and S. Singh, "ATCP: TCP for mobile ad hoc networks," IEEE Journal on Selected Areas in Communications, vol. 19, pp. 1300-1315, 2001.

[13] G. Holland and N. Vaidya, “Analysis of TCP performance over mobile ad hoc networks,” – Part II: Simulation details and results, Technical report TR99-005, 1999.

[14] Sumitha Bhandarkar, A.L. Narasimha Reddy, Mark Allman, Ethan Blanton, “Improving the Robustness of TCP to Non-Congestion Events“, RFC 4653. 2006

[15] Shugong Xu Tarek Saadawi Myung Lee, “Comparison of TCP Reno and Vegas in Wireless Mobile Ad Hoc Networks“Dept. of Electrical Engineering, City University of New York, City College, New York, NY 10031.

[16] G. Holland and N. Vaidya, "Analysis of TCP performance over mobile ad hoc networks," Wireless Networks, vol. 8, pp. 275-288, 2002.

[17] B.S. Bakshi, P. Krishna, D.K. Pradhan and N.H. Vaidya, Improving performance of TCP over wireless networks, in: International Conference on Distributed Computing Systems, 1997.

[18] J. Broch, D.A. Maltz, D.B. Johnson, Y. Hu and J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in: ACM/IEEE International Conference on Mobile Computing and Networking pp. 85–97, 1998.

[19] T. D. Dyer and R. V. Boppana, "A comparison of TCP performance over three routing protocols for mobile ad hoc networks," presented at Proceedings of the 2001 ACM International Symposium on Mobile Ad Hoc Networking and Computing: MobiHoc, 2001.

[20] H.Bakht, “A Study of Routing Protocols for Mobile Ad-hoc Networks” in 1st International Computer Engineering Conference, 2004.

[21] J. Broch, D.B. Johnson and D.A. Maltz, The dynamic source routing protocol for mobile ad hoc networks, Internet Draft, IETF MANET Working Group, 1998.

[22] H.Bakht, M.Merabti, and R.Askwith, “Mobile Ad-hoc On-demand Data Delivery Protocol,” 3rd Annual Post-Graduate Symposium on the Convergence of Telecommunications”, Networking and Broadcasting, 2002.

[23] A. Gupta and H. D. Sharma, "A Survey on Wireless Ad Hoc Networks," IETE Technical Review, vol. 20, pp. 339-347, 2003.

[24] T. -Y. Wu, C. -Y. Huang, and H. -C. Chao, "A survey of mobile IP in cellular and mobile ad-hoc network environments," Ad Hoc Networks, vol. 3, pp. 351-370, 2005.

[25] S. Toumpis and A. J. Goldsmith, "Capacity regions for wireless ad hoc networks," IEEE Transactions on Wireless Communications, vol. 2, pp. 736-748, 2003.

[26] H.Bakht. “Critical ad-hoc networking features”, Published in Computing Unplugged, 2005.

[27] C.E. Perkins, E.M. Royer, “Ad hoc on demand distance vector (AODV) routing, Internet Draft”, Mobile Ad Hoc Network (MANET) Working Group, IETF, 1998.

[28] S Pradhan, E Lawrence and J Das, S Newton "Bluetooth potential in the m-enterprise: a feasibility study", Information Technology: Coding and Computing, 2004

[29] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," Wireless Communications and Mobile Computing, vol. 2, pp. 483-502, 2002.

[30] H.Bakht. "Some characteristics of Mobile Ad-Hoc Network", Published in Computing Unplugged, July 2004.

[31] W. A. Melendez, E. L. Petersen 'The upper layers of the ISO/OSI reference model (Part II)' Computer Standards & Interfaces pp.185-199. 1999.

[32] Shin-Jer Yang , Yung-Chieh Lin, "Tuning Rules in TCP Congestion Control on the Mobile Ad Hoc Networks", Proceedings of the 20th International Conference on Advanced Information Networking and Applications, Volume 01 , 2006.

[33] Christian Lochert, Björn Scheuermann, Martin Mauve, "A survey on congestion control for mobile ad hoc networks", Wireless Communications & Mobile Computing, Volume 7 , Issue 5, pp. 655 – 676, June 2007, ISSN:1530-8669.

[34] Jehan.M, G.Radhamani, T.Kalakumari, "A survey on congestion control algorithms in wired and wireless networks", Proceedings of the International conference on mathematical computing and management (ICMCM 2010), June 2010.

[35] Foez ahmed, Sateesh Kumar Pradhan, Nayeema Islam, Sumon Kumar Debnath,"Performance Evaluation of TCP over Mobile Ad-hoc Networks" in (IJCSIS) International Journal of Computer Science and Information Security ,Vol. 7, No. 1, 2010.

[36] K. Satyanarayan Reddy and Lokanatha C. Reddy, "A survey on congestion control mechanisms in high speed networks", IJCSNS-International Journal of Computer Science and Network Security, vol. 8, no. 1, pp. 187 – 195, 2008.

[37] Lianghui Ding, Wenjun Zhang, Hui Yu, Xinbing Wang, Youyun Xu, "Incorporating TCP acknowledgements in MAC layer in IEEE 802.11 multihop ad hoc networks", GLOBECOM'09 Proceedings of the 28th IEEE conference on Global telecommunications, 2009.

[38] Lianghui Ding, Wenjun Zhang, Hui Yu, Xinbing Wang, Youyun Xu, "Improve throughput of TCP-Vegas in multihop ad hoc networks", Journal Computer Communications, Volume 31 Issue 10, pp. 2581-2588 June, 2008.

[39] Farzaneh Razavi Armaghani, Sudhanshu Shekhar Jamuar, Sabira Khatun, Mohd Fadlee A. Rasid, "Performance Analysis of TCP with Delayed Acknowledgments in Multi-hop Ad-hoc Networks", Wireless Personal Communications, Volume 56, Number 4, pp. 791-811, 2011.

[40] Elmar Schoch, Michael Feiri, Frank Kargl, Michael Weber, "Simulation of Ad Hoc Networks: ns-2 compared to JiST/SWANS", SIMUTools March 3–7, 2008.

[41] Rimon Barr, “SWANS– Scalable Wireless Ad hoc Network Simulator”, User Guide, March 19, 2004.

[42] Fard, M.A.K. “Improve TCP performance over mobile ad hoc network by retransmission timeout adjustment”, IEEE 3rd International Conference on Communication Software and Networks (ICCSN), pp. 437 – 441, 2011.

[43] Ahmad Dalal'ah, Samir Bataineh, Awos O. Kan'an, “Improving TCP performance over mobile ad hoc networks”, International Journal of Internet Technology and Secured Transactions, Vol. 2, pp. 137 – 159, 2010.

[44] Dhananjay Bisen, Sanjeev Sharma, “Improving performance of TCP-NewReno over mobile Ad-hoc Networks using ABRA”, International Journal of Wireless & Mobile Networks (IJWMN) Vol. 3, No. 2, 2011.

[45] K. Batri, S. Anbu karuppusamy, “Improving TCP Performance in Ad-Hoc Networks”, European Journal of Scientific Research, Vol.65 No.2, pp. 237-245, 2011.

[46] Javier Gomez, Luis A. Mendez, Victor Rangel, Andrew T. Campbell, “Power Controlled QoS tuning for wireless ad hoc networks PCQoS”,Telecommunication System, 2010.

[47] Monika Rani, Harish Kumar, Gurpal Singh3, “Optimal Routing Protocol for TCP-NewReno in Wireless Mobile Ad-hoc Networks”, International Journal of Network and Mobile Technologies, Vol 2, Issue 3, 2011.

[48] Md. Mohsin Ali, A. K. M. Sazzadul Alam, and Md. Shohan Sarker, “TCP Performance Enhancement in Wireless Mobile Ad Hoc Networks”, International Journal on Internet and Distributed Computing Systems. Vol 1 No: 1, 2011.

[49] F Alam, R. Askwith and M. Merabti, “Node Feedback Based TCP Scheme for Mobile Ad-hoc Network”, GESJ: Computer Science and Telecommunications 2011|No.2 (31).

[50] D.D. Clark “Window and Acknowledgement Strategy in TCP” RFC 813, July 1982.

[51] K. Ramakrishnan, S. Floyd and D. Black"The Addition of Explicit Congestion Notification (ECN) to IP" RFC 3168, September 2001.

Appendix

Programing Codes:

```
public class TransNFBTCP
{

public static class NFBTCPOptions extends TransInterface.TransMessage
 {

public NFBTCPOptions  ()
 {

}
 /**
* Returns the size of the option in a message. *
* @return size of option
*/

public int getSize()


{
return 0;
}

/**
* Retrieve the option and store it in the given byte array.
*
* @param msg byte array to copy the option to
* @param offset starting index in the destination array
*/




public void getBytes(byte[] msg, int offset)



{
throw new RuntimeException("not implemented");
}
}

/**
* Data structure for NFBTCP Message.
* NFBTCP Packet
*  header:
*      srcPort           : 2
*      dstPort           : 2
*      seqNum           : 4
*      ackNum           : 4
*      offset and flags  : 2
*      window size       : 2
*      checksum          : 2
*      urgent pointer    : 2
*  TOTAL HEADER SIZE     : 20
*  options               : variable
*  data                  : variable
*/



public static class NFBTCPMessage extends TransInterface.TransMessage
{

/**
* Minimum size of NFBTCP message.
*/



public static final int HEADER_SIZE = 20;

/**
* 16-bit source port number.
*/




private short srcPort;

/**
* 16 bit destination port number.
*/



private short dstPort;

/**
* 32 bit sequence number of first data octet in this segment.
*/




private int seqNum;

/**
* 32 bit acknowledgement number.
*/
```

```
private int ackNum;

/**
* 16 bit offset and flags.
* this contains:
*  4 bit data offset (number of 32 bit words in the NFBTCP header)
*  6 bit reserved
*  1 bit URG  - urgent pointer field significant
*  1 bit ACK  - acknowledgement field significant
*  1 bit PSH  - push function
*  1 bit RST  - reset the connection
*  1 bit SYN  - synchronize sequence numbers
*  1 bit FIN  - no more data from sender
*/



private short offsetAndFlags;

/**
* 16 bit window size (number of data octets to be accepted).
*/



private short windowSize;

/**
* checksum.
*/



private short errorChecksum;

/**
* current value of the urgent pointer as a positive offset from the sequence
number.
*/



private short urgentPointer;

/**
* options field (contains padding to make this field 32 bit boundary).
*/



private NFBTCPOptions options;

/**
```

```
* data.
*/



private Message payload;


/**
* constructor for NFBTCPMessage.
*
* @param srcPort source port number
* @param dstPort destination port number
* @param seqNum sequence number
* @param ackNum acknowledgement number
* @param offset data offset (start of data in the header; used when the packet
has options)
* @param URG urgent flag
* @param ACK acknowledgement flag
* @param PSH push flag
* @param RST reset flag
* @param SYN SYN flag
* @param FIN FIN flag
* @param windowSize size of receiving window
* @param data data
*/




public NFBTCPMessage (short srcPort, short dstPort, int seqNum, int ackNum,
short offset,
boolean URG, boolean ACK, boolean PSH, boolean RST, boolean SYN,
boolean FIN, short windowSize, Message data)

{
this.srcPort = srcPort;

this.dstPort = dstPort;

this.seqNum = seqNum;

this.ackNum = ackNum;

// filling offsetAndFlags
this.offsetAndFlags = (short)((short)(offset << 12) + (short)((URG ? 1 : 0) <<
5)
+ (short)((ACK ? 1 : 0) << 4) + (short)((PSH ? 1 : 0) << 3) +
(short)((RST ? 1 : 0) << 2) + (short)((SYN ? 1 : 0) << 1) + (short)(FIN ? 1 :
0));

this.windowSize = windowSize;

this.payload = data;
this.options = new NFBTCPOptions ();
}

/**
* constructor for NFBTCPMessage (reconstruct NFBTCPMessage from byte array).
*
* @param data array containing NFBTCP message
* @param offset start index to read the array
*/





public NFBTCPMessage (byte[] data, int offset)

{
byte[] temp;

// source port (unsigned short)
temp = new byte [2];
System.arraycopy(data, offset, temp, 0, 2);

srcPort = (short) Pickle.arrayToUShort(temp, 0);

// destination port (unsigned short)
temp = new byte [2];
System.arraycopy(data, offset+2, temp, 0, 2);

dstPort = (short) Pickle.arrayToUShort(temp, 0);

// sequence number (unsigned integer)
temp = new byte [4];
System.arraycopy(data, offset+4, temp, 0, 4);

seqNum = (int) Pickle.arrayToUInt(temp, 0);

// acknowledgement number (unsigned integer)
System.arraycopy(data, offset+8, temp, 0, 4);
ackNum = (int)Pickle.arrayToUInt(temp, 0);

// offset and flags (unsigned short)
System.arraycopy(data, offset+12, temp, 0, 2);
offsetAndFlags = (short) Pickle.arrayToUShort(temp, 0);

// window size (unsigned short)
System.arraycopy(data, offset+14, temp, 0, 2);
windowSize = (short) Pickle.arrayToUShort(temp, 0);

// checksum (short)
System.arraycopy(data, offset+16, temp, 0, 2);
errorChecksum = (short) Pickle.arrayToUShort(temp, 0);

// urgent pointer (short)
System.arraycopy(data, offset+18, temp, 0, 2);
```

```
urgentPointer = (short) Pickle.arrayToUShort(temp, 0);

// options
int tempOffset = getOffset ();

int diff = tempOffset - 5;

if (diff == 0)

{
this.options = new NFBTCPOptions ();
}

else

{
this.options = new NFBTCPOptions ();
}

// payload
temp = new byte [data.length-20-options.getSize()];
System.arraycopy(data, offset+20+options.getSize(), temp, 0,
data.length-20-options.getSize());
payload = new MessageBytes (temp);
}

/**
* Method called to create a SYN packet.
*
* @param sourcePort source port number
* @param destPort destination port number
* @param seqNumber sequence number
* @param windowSize size of receiving window
* @return SYN packet
*/

//Network formation
private static class NFMessage implements NFBTCPMessage
  {
private static final int MESSAGE_SIZE = 20;  private NetAddress ip;  private
int seqNum;

public AdhocFMessage(NetAddress ip, int seqNum)
{
      this.ip = ip; this.seqNum = seqNum;
    }
 public NetAddress getIp()
 {
      return ip;
 }
 public JAPMessage(NetAddress ip, int seqNum)

{
      this.ip = ip; this.seqNum = seqNum;
   }
 public NetAddress getIp()
 {
      return ip;
    }
public int getSize()
{
      return JAPMESSAGE_SIZE;
 }
public AdhocFMessage(NetAddress ip, int seqNum)
{
      this.ip = ip; this.seqNum = seqNum;
 }
 public NetAddress getIp()
{
      return ip;
}
public AACKMessage(NetAddress ip, int seqNum)
{
      this.ip = ip; this.seqNum = seqNum;
}
public NetAddress getIp()
{
      return ip;
 }
public int getSize()
{
      return AACKMessage_SIZE;
 }}


public static NFBTCPMessage createSYNPacket (int sourcePort, int destPort, int
seqNumber,
short windowSize)

{
int seqNum = seqNumber;

int ackNum = 0;

short offset = 5;

boolean URG = false;

boolean ACK = false;

boolean PSH = false;

boolean RST = false;

boolean SYN = true;

boolean FIN = false;

return new NFBTCPMessage ((short)sourcePort, (short)destPort, seqNum, ackNum,
offset,
URG, ACK, PSH, RST, SYN, FIN, windowSize, new MessageBytes (""));
}

/**
* Method called to create a SYNACK packet.
*
* @param sourcePort source port number
* @param destPort destination port number
* @param seqNumber sequence number
* @param ackNumber acknowledgement number
* @param windowSize size of receiving window
* @return SYNACK packet
*/

// Creating acknowledgement packet

public static NFBTCPMessage createACKPacket (int sourcePort, int destPort, int
seqNumber, int ackNumber, short windowSize)
{
int seqNum = seqNumber;
int ackNum = ackNumber;
short offset = 5;
boolean URG = false;
boolean ACK = true;
boolean PSH = false;
boolean RST = false;
boolean SYN = false;
boolean FIN = false;

return new NFBTCPMessage ((short)sourcePort, (short)destPort, seqNum, ackNum,
offset, URG, ACK, PSH, RST, SYN, FIN, windowSize, new MessageBytes (""));
}

    /**
     * Method called to create a FIN packet.
     *
     * @param sourcePort source port number
     * @param destPort destination port number
     * @param seqNumber sequence number
     * @param ackNumber acknowledgement number
     * @param windowSize size of receiving window
     * @return FIN packet
     */



//Forwarding Data Packets
private static class PrecursorSet
{
      private Map map = new HashMap();


             private RouteNFBTCP thisNode;
    /**
     * Constructs a new PrecursorSet object.
```

```
      *
      * @param thisNode reference to this RouteNFBTCP instance
      */
     public PrecursorSet(RouteNFBTCP thisNode)
     {
       this.thisNode = thisNode;
     }
     /**
      * Returns an Iterator for the set.
      *
      * Each item of the iterator is of type Map.Entry,
      * with map           keys of type MacAddress, and map values of type
PrecursorInfo
      *
      * @return the iterator
      */
   public Iterator iterator()
    {
       return map.entrySet().iterator();
     }

     /**
      * Adds an item to the precursor set.
      *
      * @param m Mac address of node to add to set
      */

 public void add(MacAddress m)
     {
printlnDebug("Adding "+m+" to precursor set", thisNode.netAddr); map.put(m,
new PrecursorInfo());
}
     /**
      * Removes an item from the precursor set.
      *
      * @param m Mac address of the node to remove from set
      */

public void remove(MacAddress m)
{
      printlnDebug("Removing "+m+" from precursor set", thisNode.netAddr);
      map.remove(m);
     }






public static NFBTCPMessage createFINPacket (int sourcePort, int destPort,
int seqNumber, int ackNumber, short windowSize)

{
int seqNum = seqNumber;
int ackNum = ackNumber;

short offset = 5;

boolean URG = false;

boolean ACK = true;

boolean PSH = false;

boolean RST = false;

boolean SYN = false;

boolean FIN = true;

return new NFBTCPMessage ((short)sourcePort, (short)destPort, seqNum, ackNum,
offset,
URG, ACK, PSH, RST, SYN, FIN, windowSize, new MessageBytes (""));
}

/**
* Method called to create a RST packet.
*
* @param sourcePort source port number
* @param destPort destination port number
* @param seqNumber sequence number
* @param ackNumber acknowledgement number
* @param windowSize size of receiving window
* @return RST packet
*/


//Link Failure Detection

private static class LinkFailureDetection (LFD) implements NFBTCPMessage
{
     private static final int MESSAGE_SIZE = 20;

    /** List of net addresses for destinations that have become unreachable. */
    private LinkedList unreachableList;

    /**
     * Constructs a new Link Failure Update Packet  Message object with an
empty unreachable list.
     */

public LinkFailureUpdatePacket ()
{
      this(new LinkedList());
 }
    /**
     * Constructs a new Route Link Update Packet  object with a given
unreachable list.
```

```
     *
     * @param list List   of net addresses for destinations that have become
unreachable
     */

public LinkUpdatePacket (LinkedList list)
{
      this.unreachableList = list;
}
    /**
     * Returns the unreachable list.
     *
     * @return linked list of unreachable node net addresses
     */

public LinkedList getUnreachableList()
{
      return this.unreachableList;
    }
    /**
     * Add an unreachable node.
     *
     * @param node netAddress of node to be added
     */

public void addUnreachable(NetAddress node)
{
      this.unreachableList.add(node);
}


//Link Capacity detection

public static NFBTCPMessage createSYNACKPacket (int sourcePort, int destPort,
int seqNumber, int ackNumber, short windowSize)

{
             int seqNum = seqNumber;

             int ackNum = ackNumber;

             short offset = 5;

             boolean URG = false;

             boolean ACK = true;

             boolean PSH = false;

             boolean RST = false;

             boolean SYN = true;

             boolean FIN = false;
```

```
            return new NFBTCPMessage ((short)sourcePort, (short)destPort,
seqNum, ackNum, offset,
        URG, ACK, PSH, RST, SYN, FIN, windowSize, new MessageBytes (""));
    }

    /**
     * Method called to create an ACK packet.
     *
     * @param sourcePort source port number
     * @param destPort destination port number
     * @param seqNumber sequence number
     * @param ackNumber acknowledgement number
     * @param windowSize size of receiving window
     * @return first ACK packet (ACK for SYNACK packet)
     */



//out of order delivery notification

public static NFBTCPMessage createRSTPacket (int sourcePort, int destPort, int
seqNumber, int ackNumber, short windowSize)

{
int seqNum = seqNumber;

      int ackNum = ackNumber;

      short offset = 5;

      boolean URG = false;

      boolean ACK = false;

      boolean PSH = false;

      boolean RST = true;

      boolean SYN = false;

      boolean FIN = true;

      return new NFBTCPMessage ((short)sourcePort, (short)destPort, seqNum,
ackNum, offset,
        URG, ACK, PSH, RST, SYN, FIN, windowSize, new MessageBytes (""));
    }



public static NFBTCPMessage createRSTPacket (int sourcePort, int destPort,
int seqNumber, int ackNumber, short windowSize)

{
int seqNum = seqNumber;
int ackNum = ackNumber;

short offset = 5;

boolean URG = false;

boolean ACK = false;

boolean PSH = false;

boolean RST = true;

boolean SYN = false;

boolean FIN = true;

return new NFBTCPMessage ((short)sourcePort, (short)destPort, seqNum, ackNum,
offset,
URG, ACK, PSH, RST, SYN, FIN, windowSize, new MessageBytes (""));
}


// Accessor functions

/**
* Accessor for source port.
*
* @return source port
*/




public short getSrcPort ()

{
return this.srcPort;
}

/**
* Accessor for destination port.
*
* @return destination port
*/




public short getDstPort ()

{
return this.dstPort;
}

/**
```

```
* Accessor for sequence number.
*
* @return sequence number
*/



public int getSeqNum ()

{
return this.seqNum;
}

/**
* Accessor for acknowledgement number.
*
* @return acknowledgement number
*/



public int getAckNum ()

{
return this.ackNum;
}

/**
* Accessor for offset in the message.
*
* @return offset from the beginning to header to data
*/



public short getOffset ()

{
return (short)(this.offsetAndFlags >> 12);
}

/**
* Accessor for URGENT flag.
*
* @return state of the URG flag (true if flag is set)
*/



public boolean getURG ()

{
return (((this.offsetAndFlags >> 5) % 2) > 0 ? true : false);
}
```

```
/**
* Accessor for ACK flag.
*
* @return state of the ACK flag (true if flag is set)
*/



public boolean getACK ()

{
return (((this.offsetAndFlags >> 4) % 2) > 0 ? true : false);
}

/**
* Accessor for PSH flag.
*
* @return state of the PSH flag (true if flag is set)
*/




public boolean getPSH ()

{
return (((this.offsetAndFlags >> 3) % 2) > 0 ? true : false);
}

/**
* Accessor for RST flag.
*
* @return state of the RST flag (true if flag is set)
*/



public boolean getRST ()

{
return (((this.offsetAndFlags >> 2) % 2) > 0 ? true : false);
}

/**
* Accessor for SYN flag.
*
* @return state of the SYN flag (true if flag is set)
*/
```

```
public boolean getSYN ()

{
return (((this.offsetAndFlags >> 1) % 2) > 0 ? true : false);
}

/**
* Accessor for FIN flag.
*
* @return state of the FIN flag (true if flag is set)
*/



public boolean getFIN ()

{
return ((this.offsetAndFlags % 2) > 0 ? true : false);
}

/**
* Accessor for window size.
*
* @return window size
*/




public short getWindowSize ()

{
return this.windowSize;
}

/**
* Accessor for options.
*
* @return NFBTCPOptions object
*/




public NFBTCPOptions getOptions ()
{
return this.options;
}

/**
* Accessor for payload.
*
* @return payload
*/
```

```
public Message getPayload ()
{
return this.payload;
}

/**
* Returns the size of the NFBTCP message.
*
* @return size of message
*/




public int getSize()

{
return HEADER_SIZE + options.getSize() + payload.getSize();
}

/**
* Retrieves the message in byte array.
*
* @param msg byte array to store the message
* @param offset start index of the destination array
*/





public void getBytes(byte[] msg, int offset)

{
// source port (unsigned short)
Pickle.ushortToArray(srcPort, msg, offset);

// destination port (unsigned short)
Pickle.ushortToArray(dstPort, msg, offset+2);

// sequence number (unsigned integer)
Pickle.uintToArray(seqNum, msg, offset+4);

// acknowledgement number (unsigned integer)
Pickle.uintToArray(ackNum, msg, offset+8);

// offset and flags (unsigned short)
Pickle.ushortToArray(offsetAndFlags, msg, offset+12);

// window size (unsigned short)
Pickle.ushortToArray(windowSize, msg, offset+14);

// checksum (short)
Pickle.ushortToArray(errorChecksum, msg, offset+16);
```

```
// urgent pointer (short)
Pickle.ushortToArray(urgentPointer, msg, offset+18);

// options
options.getBytes (msg, offset+20);

// payload
payload.getBytes (msg, offset+20+options.getSize());
}

/**
* Returns string representation of the NFBTCP message.
*
* @return string representation of the message
*/



public String toString()

{
StringBuffer sb = new StringBuffer();

sb.append("src="+getSrcPort());

sb.append(" dst="+getDstPort());

sb.append(" seq="+getSeqNum());

sb.append(" ack="+getAckNum());

sb.append(" off="+getOffset());

sb.append(" flags:");

if(getURG()) sb.append(" URG");

if(getACK()) sb.append(" ACK");

if(getPSH()) sb.append(" PSH");

if(getRST()) sb.append(" RST");

if(getSYN()) sb.append(" SYN");

if(getFIN()) sb.append(" FIN");

sb.append(" win="+getWindowSize());

String payload = new String (((MessageBytes)getPayload()).getBytes());

sb.append(" payload=("+payload.length()+") ");

if (payload.length()>10) payload=payload.substring(0, 10)+"...";
```

```
sb.append(payload);
return sb.toString();
}

/**
* Prints out the message header and payload.
*
* @param numTabs number of tabs
* @param isPrintPayload set to true to print out payload in message
*/




public void printMessage (int numTabs, boolean isPrintPayload)

{

String tabs = "";

for (int i = 0; i < numTabs; i++)

{
tabs = tabs + "\t";
}


System.out.println (tabs + "\tsrc port: " + getSrcPort() + "\tdst port: " +
getDstPort());

System.out.println (tabs + "\tseq num: " + getSeqNum());

System.out.println (tabs + "\tack num: " + getAckNum());

System.out.println (tabs + "\toffset: " + getOffset());

System.out.print (tabs + "\tflags: ");
if (getURG())

{
System.out.print ("URG ");
}

if (getACK())

{
System.out.print ("ACK ");
}

if (getPSH())

{
System.out.print ("PSH ");
}

if (getRST())

{
System.out.print ("RST ");
}

if (getSYN())

{
System.out.print ("SYN ");
}

if (getFIN())

{
System.out.print ("FIN ");
}

System.out.println ();

System.out.println (tabs + "\twindow size: " + getWindowSize());

if (isPrintPayload)

{
String temp = new String (((MessageBytes)getPayload()).getBytes());

int length = temp.length();

if (length > 10)

{
temp = temp.substring (0, 10);
temp = temp + " ...";
}

System.out.println (tabs + "\tpayload: " + temp + " (" + length + ")");
}
}

/**
* Prints out the message header with zero tabs. */




public void printMessage ()
{
printMessage (0);
}

/**
```

```
* Prints out the message header.
*
* @param numTabs number of tabs
*/




public void printMessage (int numTabs)

{
printMessage (numTabs, false);
}


} // class: NFBTCPMessage


//////////////////////////////////////////////////
// NFBTCP entity implementation
//

/**
* probability (in percent) that a message will not be sent.
* (0 --> no packets are dropped; 100 --> no packets are transmitted)
*/



private static final int DROP_PROBABILITY = 5;

/** Entity reference to itself. */



private TransInterface.TransNFBTCPInterface self;

/** Entity reference to network layer. */



private NetInterface netEntity;

/** Hashmap to hold references to socket callbacks. */



private HashMap handlers;

/**
* Constructor.
*/
```

```
public TransNFBTCP()

{self = (TransInterface.TransNFBTCPInterface)JistAPI.proxy(
this, TransInterface.TransNFBTCPInterface.class);
handlers = new HashMap ();
}

/**
* Returns an entity reference to this object.
*
* @return entity reference to TransNFBTCP itself
*/



public TransInterface.TransNFBTCPInterface getProxy()

{
return self;
}

/**
* Sets the reference to the network layer.
*
* @param netEntity entity reference to network layer
*/



public void setNetEntity(NetInterface netEntity)
{

if(!JistAPI.isEntity(netEntity)) throw new IllegalArgumentException("expected
entity");

this.netEntity = netEntity;
}

/** {@inheritDoc} */



public void addSocketHandler(int port, SocketHandler socketCallback)

{
handlers.put(new Integer (port), socketCallback);

if (NFBTCPSocket.PRINTOUT >= NFBTCPSocket.INFO)

{
System.out.println ("TransNFBTCP::addSockethandler: port = " + port);
}
}
*/
/** {@inheritDoc} */
```

```
public void delSocketHandler(int port)

{handlers.remove (new Integer (port));

if (NFBTCPSocket.PRINTOUT >= NFBTCPSocket.INFO)

{System.out.println ("TransNFBTCP::delSockethandler: port = " + port);
}
}

*/ /** {@inheritDoc} */


public boolean checkSocketHandler(int port)

{boolean ret = handlers.containsKey (new Integer (port));
return ret;
}
*/
/** {@inheritDoc} */


public void receive(Message msg, NetAddress src, MacAddress lastHop,
byte macId, NetAddress dst, byte priority, byte ttl)

{
int dstPort = ((NFBTCPMessage)msg).getDstPort();
SocketHandler handler =
(SocketHandler)handlers.get(new Integer (dstPort));

if(handler==null)

{if (NFBTCPSocket.PRINTOUT >= NFBTCPSocket.FULL_DEBUG)

{System.out.println ("%%%%%%% TransNFBTCP::receive (t=" + JistAPI.getTime()+")
-> handler for port " + dstPort + " = null!!!");

}
return;
}
JistAPI.sleep(Constants.TRANS_DELAY);

handler.receive(msg, src, ((NFBTCPMessage)msg).getDstPort());
}

*/ /** {@inheritDoc} */


public void send(Message msg, NetAddress dst, int dstPort,
int srcPort, byte priority)

{// get a random number between 0 and 100
int prob = Math.abs(Constants.random.nextInt()) % 101;

if (prob >= DROP_PROBABILITY)


{JistAPI.sleep(Constants.TRANS_DELAY);

netEntity.send(msg, dst, Constants.NET_PROTOCOL_NFBTCP,

priority, Constants.TTL_DEFAULT);
}

else


{if (NFBTCPSocket.PRINTOUT >= NFBTCPSocket.NFBTCP_DEBUG)


{System.out.println ("%%%%%% TransNFBTCP::send: PACKET DROPPED: " + msg);
}
}
}

*/

}// class: TransNFBTCP
```