Université Pierre et Marie Curie - Paris VI

A Performance Study of TCP Tahoe, Reno, New-Reno, SACK, Vegas and WestwoodNR in terms of Energy Consumption within an Ad Hoc Environment

Done by Alaa SEDDIK GHALEB
Supervised by Sidi-Mohammed SENOUCI
and Yacine GHAMRI DOUDANE

LIP6 - September 2004

Contents

CHAPTER I - Introduction

The advent of ubiquitous computing and the proliferation of portable computing devices have raised the importance of mobile and wireless networking. A **M**obile **A**d hoc **NET**work (**MANET**) is an autonomous collection of mobile nodes forming a dynamic network and communicating over wireless links [Figure 1.1]. Ad hoc communication concept allows users to communicate with each other in a multi-hop fashion without any fixed infrastructure and centralized administration. Due to their capability of handling node failures and fast topology changes, such networks are needed in situations where temporary network connectivity is required, such as in battlefields, disaster areas, and large meeting places. Such networks provide mobile users with ubiquitous communication capability and information access regardless of location.

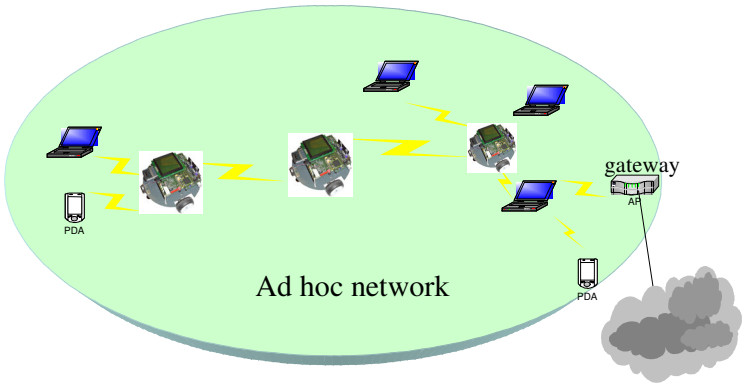


Figure 1.1 Ad-hoc Networking

TCP has gained its place as the most popular transmission protocol due to its wide compatibility to almost all today's applications. However, TCP as it exists nowadays may not well fit in mobile ad hoc networks since it was designed for wire-line networks where the channel Bit Error Rate (BER) is very low and network congestion is the primary cause of packet loss. On the contrary of wired links, wireless radio channels are affected by many factors that may lead to different levels of BER. Wireless channel behaviour cannot be predictable, but in many cases, such channels are having a high BER that cannot be neglected when studying ad hoc networks. Furthermore, node's mobility can also affect TCP sessions in ad hoc networks, which is obviously not the case of wired networks. Indeed, when nodes move, link can be broken and TCP sessions using that links can loose packets. Hence, TCP

does not have the capability to recognize whether the packet loss is due to network congestion or channel errors.

In addition to wireless channel behaviour, one of the most prominent features of Ad Hoc networks is mobility of nodes. Thus, since the devices of such a network are battery operated, they need to be energy conserving so that battery life is maximized. In the last few years, lot of research efforts have been undertaken in order to design ad hoc networking protocols that takes into consideration energy consumption aspects. Among them, a set of routing protocols [28] that have been proposed in the last few years in order to ensure network connectivity when minimizing energy consumption of mobile nodes at the same time. In the mean time, only few works dealing with energy efficiency of TCP variants have been undertaken. The objective of our work is, first, to study the performance of these TCP variants and their impact on the energy consumed by mobile nodes. Then, our second objective is to study the effect of IETF MANET routing protocols on TCP energy consumption. The result of this work can then be used, in a near future, as a guideline to design new energy-efficient TCP variant for ad hoc networks.

The remainder of this report is organized as follows: after introducing ad hoc networks and their main characteristics in chapter 1, we will list the different IETF ad hoc routing protocols in chapter 2. Chapter 3 contains an overview of recent work addressing energy efficiency and low-power design within all layers of the wireless network protocol stack. At the end of the same chapter, we will talk about TCP performance issues in ad hoc network environment. In chapter 4, we will discuss the different versions of TCP and their effect on the energy consumption of the ad hoc nodes. The effects of different ad hoc routing protocols and node's mobility on the energy consumed by TCP variants will be studied in chapter 5. Finally, we conclude our report and give some ideas for future work.

CHAPTER II - Routing in Ad Hoc Networks

2.1 Introduction

The routing protocols for ad hoc wireless network should be capable to handle a very large number of hosts with limited resources, such as bandwidth and energy. The main challenge for the routing protocols is that they must also deal with node mobility, meaning that nodes can appear and disappear in various locations. Thus, all nodes of the ad hoc network act as routers and must participate in the route discovery and maintenance of the routes to the other nodes. For ad hoc routing protocols it is essential to reduce routing messages overhead despite the increasing number of nodes and their mobility. Keeping the routing table small is another important issue, because the increase of the routing table will affect the control packets sent in the network and this in turn will cause large link overheads [37].

2.2 Classification of Ad Hoc Routing Protocols

Routing protocols are divided into two categories based on how and when routes are discovered, but both find the shortest path to the destination. Proactive routing protocols are table-driven protocols; they always maintain current up-to-date routing information by sending control messages periodically between the nodes which update their routing tables. When there are changes in the structure then the updates are propagated throughout the network. Other routing protocols are on-demand routing protocols, in other words reactive, ones which create routes when they are needed by the source node and these routes are maintained while they are needed [37]. Route construction should be done with a minimum of overhead and bandwidth consumption taking into consideration the constraint of battery lifetime. In real life systems, energy consumption is a major issue, and the routing protocols affect the energy dynamics in two ways. First, the routing overhead affects the amount of energy used for sending and receiving the routing packets, and second, the chosen routes affect which nodes will have faster decrease in energy [36]. Ad hoc routing protocols must operate in a distributed fashion allowing each node to enter and leave the network on its own, and should avoid data looping in the network. For very dynamic topologies, proactive protocols can introduce a large overhead in bandwidth and energy consumption on the network. Reactive protocols trades off this overhead with increased delay, as the route to the destination is established when it is needed based on an initial discovery

between the source and the destination [36]. Following is a summary of routing protocols studied in this work.

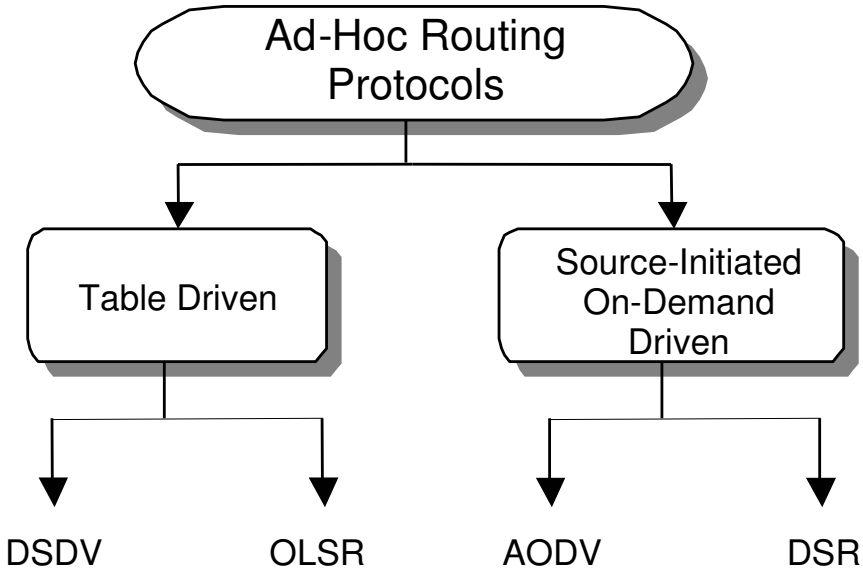


Figure 2.1. Ad hoc routing protocol categories

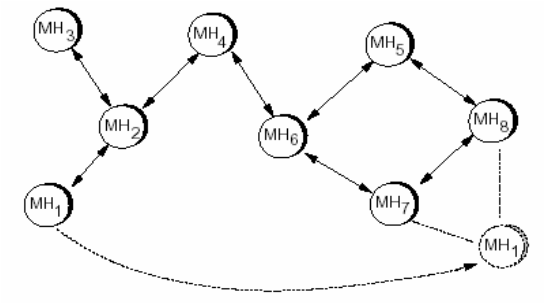
2.2.1 Table-Driven Routing Protocols

Table-driven routing protocols attempt to maintain consistent, up-to-date routing information from each node to every other node in the network. These protocols require each node to maintain one or more tables to store routing information, and they respond to changes in network topology by propagating updates throughout the network in order to maintain a consistent network view. The areas in which they differ are the number of necessary routing-related tables and the methods by which changes in network structure are broadcast. The following sections discuss some of the existing table-driven ad hoc routing protocols [30].

2.2.1.1 Destination-Sequenced Distance-Vector Routing

The Destination-Sequenced Distance-Vector Routing protocol (DSDV) described in [31] is a table-driven algorithm based on the classical Bellman-Ford routing mechanism [32]. The improvements made to the Bellman-Ford algorithm include freedom from loops in routing tables. Every mobile node in the network maintains a routing table in which all of the possible destinations within the network and the number of hops to each

destination are recorded. Each entry is marked with a sequence number assigned by the destination node. The sequence numbers enable the mobile nodes to distinguish stale routes from new ones, thereby avoiding the formation of routing loops. Routing table updates are periodically transmitted throughout the network in order to maintain table consistency. To help alleviate the potentially large amount of network traffic that such updates can generate, route updates can employ two possible types of packets. The first is known as a *full dump*. This type of packet carries all available routing information and can require multiple network protocol data units (NPDUs). During periods of occasional movement, these packets are transmitted infrequently. Smaller *incremental* packets are used to relay only that information which has changed since the last full dump. Each of these broadcasts should fit into a standard-size NPDU, thereby decreasing the amount of traffic generated. The mobile nodes maintain an additional table where they store the data sent in the incremental routing information packets. New route broadcasts contain the address of the destination, the number of hops to reach the destination, the sequence number of the information received regarding the destination, as well as a new sequence number unique to the broadcast [31]. The route labelled with the most recent sequence number is always used. In the event that two updates have the same sequence number, the route with the smaller metric is used in order to optimize (shorten) the path. Mobiles also keep track of the settling time of routes, or the weighted average time that routes to a destination will fluctuate before the route with the best metric is received (see [31]). By delaying the broadcast of a routing update by the length of the settling time, mobiles can reduce network traffic and optimize routes by eliminating those broadcasts that would occur if a better route was discovered in the very near future [30].



Every node keeps a route table (Destination-address, Metric, Sequence-number) for every possible destination

Destination	Metric	Sequence number
MH_1	2	S406_MH_1
MH_2	1	S128_MH_2
MH_3	2	S564_MH_3
MH_4	0	S710_MH_4
MH_5	2	S392_MH_5
MH_6	1	S076_MH_6
MH_7	2	S128_MH_7
MH_8	3	S050_MH_8

Advertised route table by MH_4

Destination	Metric	Sequence number
MH_4	0	S820_MH_4
MH_1	3	S516_MH_1
MH_2	1	S238_MH_2
MH_3	2	S674_MH_3
MH_5	2	S502_MH_5
MH_6	1	S186_MH_6
MH_7	2	S238_MH_7
MH_8	3	S160_MH_8

MH_4 advertised table (updated)

2.2.1.2 Optimized Link State Routing Protocol (OLSR)

The Optimized Link-State Routing Protocol (OLSR) is a proactive link-state routing protocol, employing periodic message exchange to update topological information in each node in the network, so the routes are always immediately available when needed. While having some commonalities with OSPF, OLSR is specially designed to operate in the context of ad hoc networks, i.e. in bandwidth-constrained, dynamic networks. Conceptually, OLSR contains three generic elements: a mechanism for neighbour sensing, a mechanism for efficient diffusion of control traffic, and a mechanism for selecting and diffusing sufficient topological information in the network in order to provide optimal routes [38].

2.2.1.2.1 Neighbour Sensing

Basically, neighbour sensing is the process through which a node detects changes to its neighbourhood. The neighbourhood of a node, *a*, contains the set of nodes with which there exists a direct link over which data may be transmitted (in either or both directions). Further attributes can be associated with such a link, depending on the direction(s) in which communication is possible. If traffic can only flow in one direction (e.g. if the nodes have asymmetric transmitters), the link is said to be asymmetric. If traffic can flow in both directions, the link is said to be symmetric. If there exist a symmetric link between node *b* and node *a*, node *b* is said to be a symmetric neighbour of node *a* (and vice versa). In OLSR, the concept of a two-hop neighbour is introduced. A two-hop neighbour of node *a* is simply a

node which has a symmetric link to a symmetric neighbour of node *a* AND which is not node *a* itself (i.e. node *a* can not be a two-hop neighbour of itself). A prime goal for OLSR is to be completely independent of the underlying link-layer being used. While additional information from the link layer, such as information about existence of links to neighbour nodes and link quality, may be utilized by the protocol, care is taken such that the protocol can function without. The advantages are that the protocol immediately can be deployed on most existing and anticipated wireless network interfaces and operating systems. The neighbour sensing mechanism in OLSR is designed to operate independently in the following way: each node periodically emits a HELLO-message, containing the node's own address as well as the list of neighbours known to the node, including the status of the link to each neighbor (e.g. symmetric or asymmetric). Upon receiving HELLO-messages, a node can thus gather information describing its neighbourhood and two-hop neighbourhood, as well as detect the "quality" of the links in its neighbourhood: the link from a node *a* to a neighbor *b* is symmetric if the node a sees its own address if in the HELLO-message from *b* (with any link status) - otherwise the link is asymmetric. Each node maintains an information set, describing the neighbours and the two-hop neighbours. Such information is considered valid for a limited period of time, and must be refreshed periodically to remain valid. Expired information is purged from the neighbor- and two-hop neighbor sets.

2.2.1.2.2 Generic Message Diffusion

HELLO-messages are exchanged between neighbours only. They provide each node with topological information up to two hops away. However, since ad hoc networks can be of arbitrary size, a method is required for diffusing topological information into the entire network. In OLSR, this is introduced in form of a generic way of efficiently diffusing arbitrary control traffic to all nodes in the network. While being directly used in OLSR for diffusion of topological information, the mechanism is build as an independent and efficient MPR-flooding mechanism, and may thus be used to carry other types of control traffic (e.g. for service discovery protocols etc). Due to limited bandwidth resources, the overhead from control traffic should be kept at a minimum. This, for a control message destined to all nodes in the network, implies that (i) all nodes ideally receive the message, while (ii) that not too many duplicate retransmissions of the message occurs. A simple pure flooding strategy, where all nodes forward a flooded message if they have not previously forwarded the message meets the first part of the requirement: that all nodes, ideally, receive a copy of the message. However, a given node might be receiving the same message from two neighbouring nodes.

This is illustrated in Figure 2.2.a. The fact that a message is likely to be received by a node more than once is a problem: using pure flooding, when a message is transmitted over the wireless medium, all other nodes within radio range of the transmitting node will either have to remain silent, or may experience message loss due to collisions. The OLSR protocol applies an optimized flooding mechanism, called MPR-flooding, to minimize the problem of duplicate reception of message within a region. The optimization is performed in the following way: a node selects a subset of its symmetric neighbours, called the nodes Multipoint Relays (MPR's). Each node thus has a (possibly empty) set of MPR selectors (neighbours, which have selected the node as MPR). A node, selected as MPR, has the responsibility of relaying flooded messages from its MPR selectors. A message emitted by node *a* is thus only retransmitted by node *b* if node *a* is in the MPR selector set of node *b*. As illustrated in Figure 2.2.b, "careful" selection of MPRs (the filled nodes) may greatly reduce duplicate retransmissions. While selecting MPRs, a node utilizes information describing the two-hop neighbours, as acquired from the neighbor sensing process. All nodes select their MPRs independently, possibly choosing different algorithms for selecting a "minimal" MPR set. The invariant for the algorithms is that a message, emitted by the node and relayed by its MPRs, would reach all the node's two-hop neighbours. An analysis of MPR selection algorithms can be viewed as followed: A node is informed of its MPR selector set through information piggy bagged to the HELLO-messages.

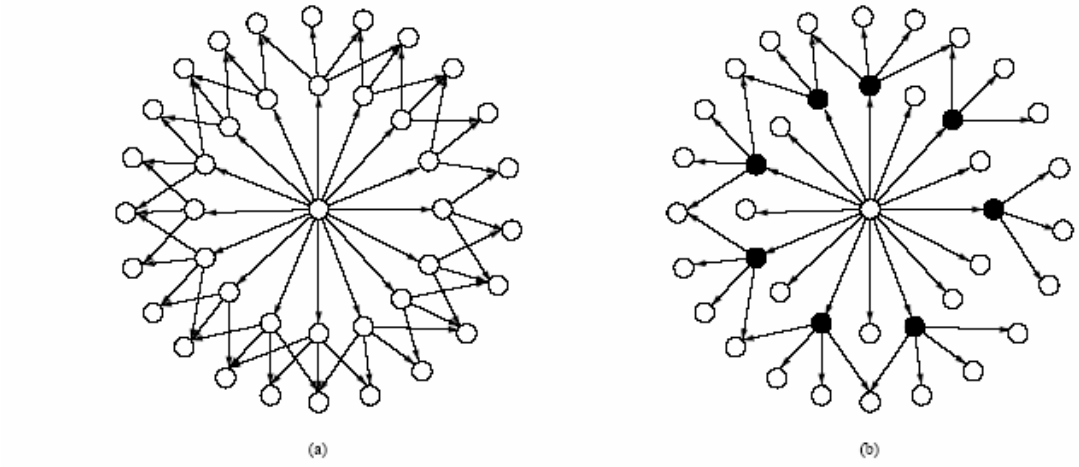


Figure 2.2. (a) Example of pure flooding, (b) diffusion using Multipoint Relays.

The source of the message is the node in the center. Each arrow pointing to a node indicates that the node receives a copy of the message. The filled nodes are selected by the center node as Multipoint Relay.

Thus when using MPR-flooding, the forwarding rule for handling flooded control messages in each node being:

1. The message must be meant to be forwarded (indicated by information in the header of the message),
2. The message must not have been received by the node before, and
3. The node must have been selected as MPR by the node, from which the message was received

The OLSR protocol specification defines a generic message format and an algorithm for processing such messages. This includes time-to-live considerations, sequence numbers, etc.

2.2.1.2.3 Topology Information

The MPR flooding mechanism is directly used by OLSR for diffusing topological information to the network. In OLSR, all nodes with a non-empty MPR selector set periodically generate a topology control message (TC-message). This TC-message is diffused to all nodes in the network, using MPR flooding. A TC-message contains the address of the node generating the TC-message, as well as the addresses of all the MPR selectors of that node. Thus through a TC-message, a node effectively announces reachability to all its MPR selectors. Since all nodes have selected an MPR set, reachability to all nodes will be announced through the network. The result is that all nodes will receive a partial topology graph of the network, made up by all reachable nodes in the network and the set of links between a node and its MPR selectors. Using this partial topology graph, it is possible to apply a shortest path algorithm for computing optimal routes from a node to any reachable destination in the network. A noticeable result is that the shortest path obtained from the partial topology yielded by the TC-messages has the same length as the shortest path from the full topology. The topological information in each node is valid for a limited period of time, and must be refreshed periodically to remain valid. To improve reactiveness to network dynamics, additional TC-messages may be generated. Expired information is purged from the topology graph.

2.2.2 Source-Initiated On-Demand Routing

A different approach from table-driven routing is source-initiated on-demand routing. This type of routing creates routes only when desired by the source node. When a node requires a route to a destination, it initiates a route discovery process within the network. This process is completed once a route is found or all possible route permutations have been examined. Once a route has been established, it is maintained by a route maintenance procedure until either the destination becomes inaccessible along every path from the source or until the route is no longer desired [30].

2.2.2.1 Ad Hoc On-Demand Distance Vector Routing

The Ad Hoc On-Demand Distance Vector (AODV) routing protocol described in [33] builds on the DSDV algorithm previously described. AODV is an improvement on DSDV because it typically minimizes the number of required broadcasts by creating routes on a demand basis, as opposed to maintaining a complete list of routes as in the DSDV algorithm. The authors of AODV classify it as a *pure on-demand route acquisition* system, since nodes that are not on a selected path do not maintain routing information or participate in routing table exchanges [33]. When a source node desires to send a message to some destination node and does not already have a valid route to that destination, it initiates a *path discovery* process to locate the other node. It broadcasts a route request (RREQ) packet to its neighbours, which then forward the request to their neighbours, and so on, until either the destination or an intermediate node with a "fresh enough" route to the destination is located. Figure 2.3.a illustrates the propagation of the broadcast RREQs across the network. AODV utilizes destination sequence numbers to ensure all routes are loop-free and contain the most recent route information. Each node maintains its own sequence number, as well as a broadcast ID. The broadcast ID is incremented for every RREQ the node initiates, and together with the node's IP address, uniquely identifies an RREQ. Along with its own sequence number and the broadcast ID, the source node includes in the RREQ the most recent sequence number it has for the destination. Intermediate nodes can reply to the RREQ only if they have a route to the destination whose corresponding destination sequence number is greater than or equal to that contained in the RREQ. During the process of forwarding the RREQ, intermediate nodes record in their route tables the address of the neighbour from which the first copy of the broadcast packet is received, thereby establishing a reverse path. If additional copies of the same RREQ are later received, these packets are discarded. Once the RREQ reaches the

destination or an intermediate node with a fresh enough route, the destination/ intermediate node responds by unicasting a route reply (RREP) packet back to the neighbour from which it first received the RREQ (Figure 2.3.b). As the RREP is routed back along the reverse path, nodes along this path set up forward route entries in their route tables which point to the node from which the RREP came. These forward route entries indicate the active forward route. Associated with each route entry is a route timer which will cause the deletion of the entry if it is not used within the specified lifetime. Because the RREP is forwarded along the path established by the RREQ, AODV only supports the use of symmetric links. Routes are maintained as follows. If a source node moves, it is able to reinitiate the route discovery protocol to find a new route to the destination. If a node along the route moves, its upstream neighbour notices the move and propagates a *link failure notification* message (an RREP with infinite metric) to each of its active upstream neighbours to inform them of the erasure of that part of the route [33]. These nodes in turn propagate the *link failure notification* to their upstream neighbours, and so on until the source node is reached. The source node may then choose to reinitiate route discovery for that destination if a route is still desired. An additional aspect of the protocol is the use of *hello* messages, periodic local broadcasts by a node to inform each mobile node of other nodes in its neighbourhood. Hello messages can be used to maintain the local connectivity of a node. However, the use of hello messages is not required. Nodes listen for retransmission of data packets to ensure that the next hop is still within reach. If such a retransmission is not heard, the node may use any one of a number of techniques, including the reception of hello messages, to determine whether the next hop is within communication range. The hello messages may list the other nodes from which a mobile has heard, thereby yielding greater knowledge of network connectivity [30].

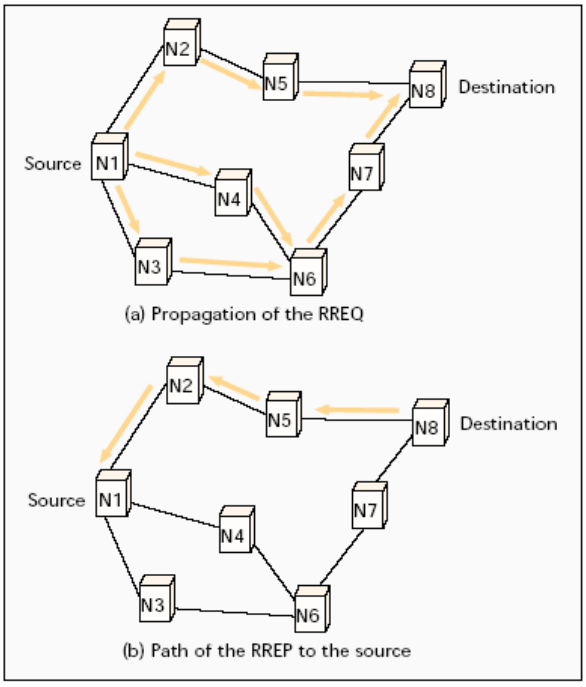


Figure 2.3. AODV route discovery

Because the AODV protocol is a flat routing protocol it does not need any central administrative system to handle the routing process. Reactive protocols like AODV tend to reduce the control traffic messages overhead at the cost of increased latency in finding new routes. In addition, AODV tries to keep the overhead of the messages small. If any node has the route information in the Routing Table about active routes in the network, then the overhead of the routing process will be minimal. The AODV has great advantage in overhead over simple protocols which need to keep the entire route from the source host to the destination host in their messages. The RREQ and RREP messages, which are responsible for the route discovery, do not increase significantly the overhead from these control messages. AODV reacts relatively quickly to the topological changes in the network and updating only the hosts that may be affected by the change, using the RRER message. The Hello messages, which are responsible for the route maintenance, are also limited so that they do not create unnecessary overhead in the network. The AODV protocol is a loop free and avoids the counting to infinity problem, which were typical to the classical distance vector routing protocols, by the usage of the sequence numbers [37].

2.2.2.2 Dynamic Source Routing

The Dynamic Source Routing (DSR) protocol presented in [34] is an on-demand routing protocol that is based on the concept of source routing. Mobile nodes are required to maintain route caches that contain the source routes of which the mobile is aware. Entries in the route cache are continually updated as new routes are learned. The protocol consists of two major phases: route discovery and route maintenance. When a mobile node has a packet to send to some destination, it first consults its route cache to determine whether it already has a route to the destination. If it has an unexpired route to the destination, it will use this route to send the packet. On the other hand, if the node does not have such a route, it initiates route discovery by broadcasting a *route request* packet. This route request contains the address of the destination, along with the source node's address and a unique identification number. Each node receiving the packet checks whether it knows of a route to the destination. If it does not, it adds its own address to the *route record* of the packet and then forwards the packet along its outgoing links. To limit the number of route requests propagated on the outgoing links of a node, a mobile only forwards the route request if the request has not yet been seen by the mobile and if the mobile's address does not already appear in the route record. A *route reply* is generated when the route request reaches either the destination itself, or an intermediate node which contains in its route cache an unexpired route to the destination [35]. By the time the packet reaches either the destination or such an intermediate node, it contains a route record yielding the sequence of hops taken. Figure 2.4.a illustrates the formation of the route record as the route request propagates through the network. If the node generating the route reply is the destination, it places the route record contained in the route request into the route reply. If the responding node is an intermediate node, it will append its cached route to the route record and then generate the route reply. To return the route reply, the responding node must have a route to the initiator. If it has a route to the initiator in its route cache, it may use that route. Otherwise, if symmetric links are supported, the node may reverse the route in the route record. If symmetric links are not supported, the node may initiate its own route discovery and piggyback the route reply on the new route request. Figure 2.4.b shows the transmission of the route reply with its associated route record back to the source node. Route maintenance is accomplished through the use of route error packets and acknowledgments. *Route error* packets are generated at a node when the data link layer encounters a fatal transmission problem. When a route error packet is received, the hop in error is removed from the node's route cache and all routes containing the hop are truncated at that point. In addition to route error messages, acknowledgments are used to verify the correct operation of the route

links. Such acknowledgments include passive acknowledgments, where a mobile is able to hear the next hop forwarding the packet along the route.

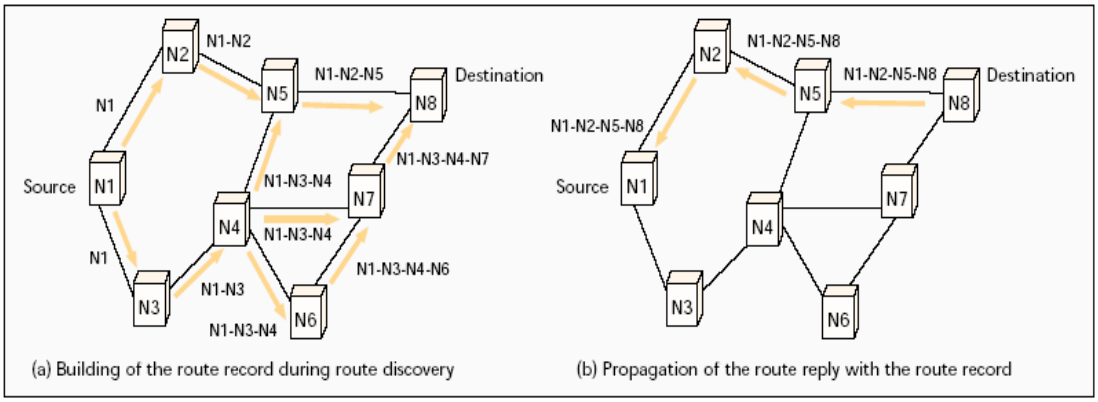


Figure 2.4. Creation of the route record in DSR

DSR has the advantage that no routing tables must be kept to route a given packet, since the entire route is contained in the packet header. The caching of any initiated or overheard routing data can significantly reduce the number of control messages being sent, reducing overhead. The primary disadvantages of DSR are that DSR is not scalable to large networks, and that it requires significantly more processing resources than most other protocols. In order to obtain routing information, each node must spend much more time processing any control data it receives, even if it is not the intended recipient [36].

2.3 Conclusion

Ad hoc routing protocols are an important issue in ad hoc networks as they must have special characteristics to cope with such networks. They must be reliable and conserve the rare network resources at the same time. The choice of the routing protocol will affect significantly the consumption of network limited resources (energy and bandwidth). The study of that effect on bandwidth is out of scope of our work. Here, we will focus our study on how they affect energy consumption in ad hoc networks.

CHAPTER III - Energy Aspects in Ad Hoc Networks

3.1 Introduction

Significant power savings may result from incorporating low-power strategies into the design of network protocols used for data communication. All the layers of the protocol stack were the subject of proposals incorporating energy conservation [24]. This section summarizes some of these proposals.

3.2 Energy Consumption in OSI Layers

3.2.1 Physical Layer

In the past, energy efficient and low-power design research has centered on the physical layer due to the fact that the consumption of power in a mobile computer is a direct result of the system hardware [24]. Several technologies are being developed to achieve low-power consumption in the hardware layer by increasing the battery capacity and reducing the energy consumption for the CPU, user interface and storage for the devices.

3.2.2 MAC Sublayer

The MAC (Media Access Control) is a sublayer of the data link layer. It interfaces with the physical layer and is represented by protocols that define how the shared wireless channels are to be allocated among a number of mobiles. One fundamental task of the MAC protocol is to avoid collisions so that two interfering nodes do not transmit at the same time. Collisions should be eliminated as much as possible since they result in retransmissions, which lead to unnecessary power consumption. There are many MAC protocols: IEEE 802.11, EC-MAC, and PAMAS. For example, power conservation in **PAMAS** (**P**ower **A**ware **M**ulti-**A**ccess **P**rotocol with **S**ignalling) is achieved by requiring mobiles that are not able to receive and send packets to turn off their wireless interface for a period of time.

3.2.3 LLC Sublayer

The logical link control (LLC) sublayer has the error control functionality. The two most important techniques used for error control are **ARQ** (**A**utomatic **R**epeat **R**equest) and **FEC** (**F**orward **E**rror **C**orrection). These two methods consume power resources due to

retransmission of data packets and greater message overhead necessary in error correction. Recent research has addressed low-power error control and several energy efficient link layer protocols have been proposed [24].

3.2.4 Network Layer

The type of routing protocol affects the energy dynamics in two ways: (i) the routing overhead affects the amount of energy used for sending and receiving the routing packets, and (ii) the chosen routes affects which nodes will have a faster decrease in energy. Hence, [28] has compared two proactive protocols (**DSDV** - **D**ynamic destination-**S**equenced **D**istance **V**ector and **OLSR** - **O**ptimized **L**ink **S**tate **R**outing) and two reactive protocols (**DSR** - **D**ynamic **S**ource **R**outing and **AODV**) in terms of energy consumption. It carried out a set of simulations, including different parameters: node speed, node number, and traffic pattern.

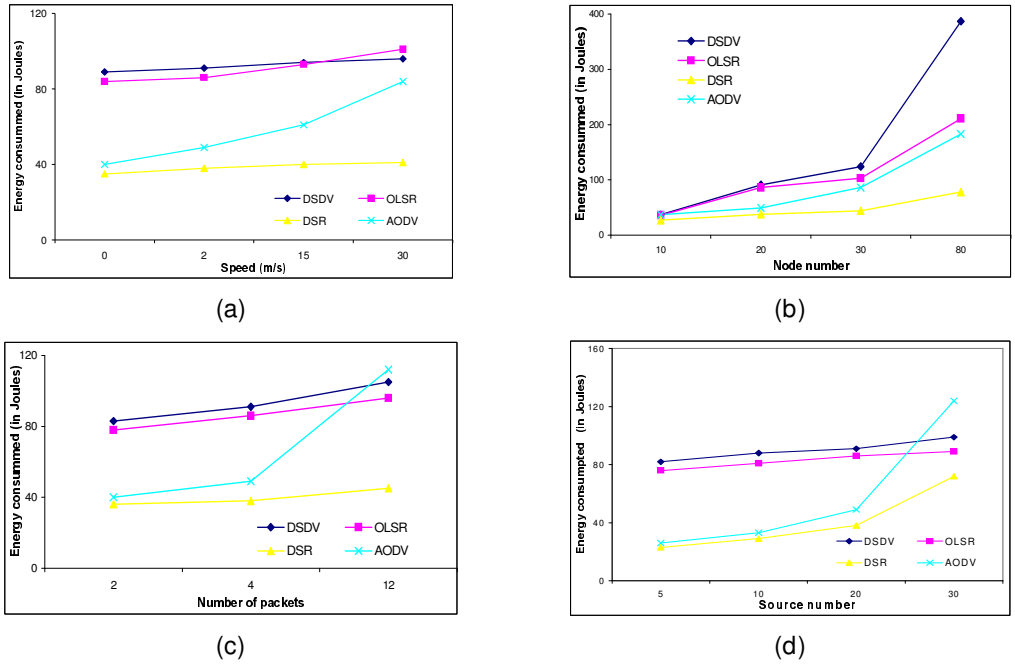


Figure 3.1 Energy consumption as a function of: (a) node speed (b) node number (c) packet rate (d) source number

The effect of node speed is shown in Figure 3.1 (a). This experiment indicates that reactive protocols use less energy than proactive protocols. Reactive protocols do not do any routing when there is no traffic in the network, whereas proactive protocols are constantly consuming energy by computing routes even when no data will be sent. We can see in Figure 3.1 (b), that

reactive protocols outperform against proactive protocols when the node number grows. As the number of nodes grows, so proactive protocols suffer from their constant updates. Thus, proactive protocols have a scalability problem. Finally, Figure 3.1 (c) and Figure 3.1 (d) show a similar behavior since the varying parameters concern the traffic. As the traffic increases, DSDV and OLSR outperform AODV. Indeed, AODV cumulatively acquires two characteristics, which are drawbacks in this context: periodical "hello messages" and frequent unnecessary route discoveries.

These conventional ad hoc routing protocols work towards optimal routes in terms of delay, which mostly result in the shortest path. This would mean that nodes with higher node degree might "die" soon since they are being used in most cases. Therefore, several routing schemes that take the power constraint into consideration for choosing the appropriate route have been proposed [25] [26] [27]. These schemes establish routes that ensure that all nodes equally deplete their battery power.

3.2.5 Transport Layer

TCP is a connection oriented transport layer protocol that provides reliable, in-order delivery of data to the TCP receiver. TCP was developed to be deployed in wired network environments. It is now considered as the default transport protocol for the new generation of wireless networks. This arises from the fact that TCP will guarantee the interoperability between Ad Hoc and wired networks. The problem with the adaptation of TCP to wireless networks is because wireless links often suffer from high BER and broken connectivity. This leads to a large number of retransmissions that unnecessarily consume battery energy. A range of schemes, *Reno* and *New Reno* for example, has been proposed to improve performance of transport mechanisms, in particular TCP, on wireless networks. Although some of these schemes led to great energy efficiency, they were not specially conceived to reduce the energy consumption at the transport layer. Our work here consists of analyzing the performance of different TCP variants in terms of energy consumption in order to find the most appropriate one of them for the mobile ad hoc network environments.

3.2.6 Application Layer

Energy efficiency at the application layer is becoming an important area of research. For example, APIs are being developed to assist software developers in creating programs that are

more power conserving. The impact of power efficiency on database systems, multimedia processing and transmission, is also considered by some researchers [24].

3.3 Energy Consumption due to L3/L4 Interaction

Since the compatibility with Internet is often desired, most mobile ad hoc implementations are designed with the TCP/IP stack in mind. Thus, in this section we will discuss the energy consumption due to the TCP/IP stack operations.

The computational energy cost of a TCP/IP session established by a wireless device can be viewed as the total energy cost of the session minus the cost of the radio and the idle energy cost of the connection (i.e. when the node is idle waiting ACKs or data segments [8][21].

To better understand which TCP/IP functions cost more energy, the computational energy cost can be decomposed as follows:

- The cost of moving data from the user space into kernel space denoted as *user –to kernel copy*. Note that this cost can be eliminated by using zero-copy [9], if available.
- The cost of copying the packets to the network interface card that denoted as *kernel-to-NIC copy*.
- The cost of processing in the TCP/IP protocol stack (TCP processing cost) which includes:
  - The cost of computing the checksum per packet (at sender and receiver),
  - The cost of ACKs (at the receiver),
  - The cost of responding to timeout events (TO) (at the sender this is the processing cost plus *kernel-to-NIC copy* cost of the retransmitted packet),
  - The cost of responding to triple duplicate ACKs (TD) (at the sender this is the processing cost plus the *kernel-to-NIC copy* cost of the retransmitted packet) and
  - Other processing costs such as window maintenance (at sender on receiving ACKs), estimate round-trip time (RTT) (at sender), interrupt handling, and timer maintenance.

Using the above decomposition, the total computational cost of a TCP session at the sender, in which *D* bytes of data are transmitted, can be written as follows:

E(D, MTU size) = *user-to-kernel copy* for D bytes + checksum cost for D/MTU packets
+ ACK cost + *kernel-to-NIC copy* cost for D/MTU packets of size MTU
+ Number of TOs * TO copy and processing cost
+ Number of TDs * TD copy and processing cost + other processing costs

Note that the total energy *E* is written as function of the MTU (Maximum Transmission Unit) size because different MTU sizes result in different total energy costs (as will be discussed in the next section). The equation at the receiver can be deduced similarly.

Actually, the two copy costs are similar at the sender and at the receiver. However, the TCP processing cost is somewhat different. At the sender the TCP processing cost includes checksum, ACK processing, congestion window updates, timeout (TO) processing, triple duplicate (TD) processing, timer processing, RTO computation, and other OS related costs. At the receiver, TCP processing includes ACK generation, checksum computation, data sequencing and receive window management. Thus, transmitting requires more energy than receiving (Fig 3.2). B. Wang and S. Singh [8] showed that TCP processing cost is not high, and is not the bottleneck to achieving high data rates, and that 60-70% of the energy cost (for transmission or reception) is accounted for by the kernel-to-NIC copy operation. Of the reminder, ~15% is accounted for in the copy operation from user space to kernel space with the remaining 15% being accounted for by TCP processing cost, and that the cost of computing checksums accounts for 20-30% of TCP processing cost.

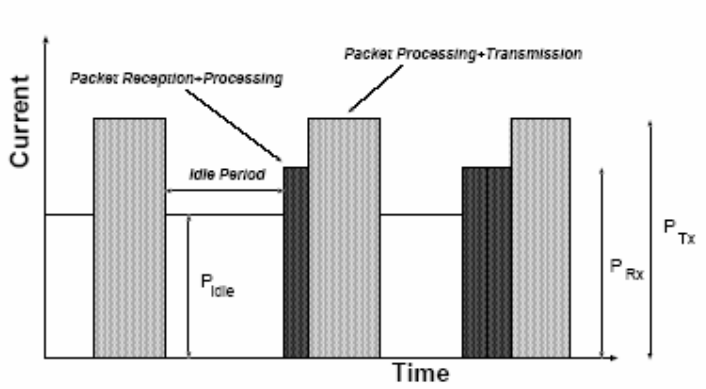


Figure 3.2 Simplified Energy Consumption Profile

The energy consumed by an interface depends on its operating mode: in the sleep state, an interface can neither transmit nor receive, so it consumes very little energy. To be able to transmit or receive, an interface must explicitly transition to the idle state, which requires both time and energy. In the idle state, an interface can transmit or receive data at any time, but it

consumes more energy than it does in the sleep state, due to the number of circuit elements that must be powered [24]. It can be seen also, that the transmission process is more expensive than the reception one.

3.4 TCP Performance Issues in Ad Hoc Networks

Mobile Ad Hoc wireless networks have been proposed as the networking solution for those situations where the network set up time is a major constraint and/or a networking infrastructure is either not available or not desirable. Ad Hoc networks allow mobile devices to exchange information using their wireless infrastructure without the need of the fixed infrastructure and the attached specialized devices commonly found in wired networks such as routers, switches, gateways, etc. As a result, every device in ad hoc wireless network can take the role of an end system, a server, a router, gateway, etc., or all of them at the same time. It is expected that the performance of TCP will be affected considerably in ad hoc networks not only due to the effects of the wireless environment but also due to specific issues only found in ad hoc networks like mobility, routing, and energy constrains [16]. In mobile networks, such as ad hoc networks, TCP displays some undesirable patterns of behavior in the context of efficient energy expenditure because of its reliability feature [15]. For example, high channel delays in a mobile network causing the TCP timer to expire will force TCP to unnecessarily retransmit the delayed packet and to consume more time and energy resulting in network performance degradation. The performance of mobile ad hoc networks will depend on many factors such as node mobility model, traffic pattern, network topology, obstacle positions, and so on. To better understand the effect of these factors, we will classify them into two categories: factors inherited from the wireless environment, and factors due to the mobile ad hoc characteristics itself. In the following, we will present some of the most important issues in a mobile ad hoc network.

3.4.1 Factors due to the Wireless Environment:

3.4.1.1 The Effect of High Bit Error Rate (BER)

Bit errors cause packets to get corrupted which result in lost TCP data segments or acknowledgements. When acknowledgements do not arrive at the TCP sender within a short amount of time (the retransmit timeout or RTO), the sender retransmits the segment,

exponentially backs off its retransmit timer for the next retransmission, reduces its congestion control window threshold, and closes its congestion window to one segment. Repeated errors will ensure that the congestion window at the sender remains small resulting in low throughput [5]. While the appropriate behavior in such situation is to simply retransmit lost packets without shrinking the congestion window to avoid the delay and the energy consumed to increase the congestion window size. It is important here to mention that, in the case of CSMA-CA (802.11) based networks; there will be of course some kind of losses. But on the other hand, there will be an error correction algorithm, meaning that not all the packets will be lost. In fact, some of hem will be corrected and resent again. This correction action will, of course, introduce some delay in the network and in the mean time will waste some of the bandwidth resource. Obviously, the delay introduced will increase the RTT value resulting in poor throughput and high-energy consumption.

3.4.1.2 The Effect of Multi-Path Routing

Some routing protocols maintain multiple routes between source and destination pairs, the purpose of which is to minimize the frequency of route recomputaion. Unfortunately, this sometimes results in a significant number of out-of sequence packets arriving at the receiver. The effect of this is that the receiver generates duplicate ACKs that cause the sender (on receipt of three *duplicate ACKs*) to invoke congestion control [5].

3.4.2 Factors due to the Mobile Ad Hoc Environment:

3.4.2.1 The Effect of Route Recomputation

When an old route is no longer available, the network layer at the sender attempts to find a new route to the destination. It is possible that discovering a new route may take significantly longer than the retransmission timeout interval (RTO) at the sender. As a result, the TCP sender times out, retransmits a packet and invokes congestion control. Thus, when a new route is discovered, the throughput will continue to be small for some time because TCP at the sender grows its congestion window using the slow start and congestion avoidance algorithm. This is clearly undesirable behaviour because the TCP connection will be very inefficient. If we imagine a network in which route computations are done frequently (due to high node mobility), the TCP connection will never get an opportunity to transmit at the maximum negotiated rate (the congestion window will

always be significantly smaller than the advertised window size from the receiver) [5]. Thus, it will be better to stop transmitting and resume it when a new route has been found.

3.4.2.2 The Effect of Network Partitions

It is likely that the ad hoc network may periodically get partitioned for several seconds at a time. 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. If the partition lasts for a significant amount of time (say several times longer than the RTO), the situation gets even worse because of phenomena called *serial timeouts*. A serial time out is a condition wherein multiple consecutive retransmissions of the same segment are transmitted to the receiver while it is disconnected from the sender. All these retransmissions are thus lost. Since the retransmission timer at the sender is doubled with each unsuccessful retransmission attempt (until it reaches 64 sec), several consecutive failures can lead to inactivity lasting one or two minutes even when the sender and receiver get reconnected [5]. And the appropriate solution here is to stop the transmission (to avoid flooding the network with packets that cannot be delivered anyway) until that the sender get reconnected to the receiver.

3.4.3 Other Factors that Affect Ad Hoc Networks:

Additionally, there are some other factors that can affect the mobile ad hoc network performance in different ways. Even though they are not directly involved in the data transmission they must be considered when studying ad hoc networks, because of their effects on data transmission [4]:

3.4.3.1 Mobility

When nodes move around there is a possibility that they will move out of range of old neighbors or into range of new ones. When that happens the ad hoc routing protocol may have to find new routes to be able to maintain communication that involved these nodes. Usually, a broken route results in performance degradation, since no data can be exchanged. To overcome this problem the network layer should find a new route as quickly as possible to resume the dropped communication. In fact, high mobility is not always a bad thing for ad hoc networks. Some authors have observed that mobility can increase performance by distributing traffic more evenly over the network.

3.4.3.1 Topology and Environment

Where the nodes are located and the nature of the surrounding environment determines which nodes can contact each other and the amount of interference from other nodes. If the nodes are located close to each other, there will be a greater chance that the data will not have to make as many hops as in a network where the nodes at further apart. On the other hand, networks with a dense concentration of nodes will experience more contention for the available capacity and also more interference. The environment affects the performance in a similar way. Actually, walls and other objects that hinder radio transmissions will lower the effect of high node density. It is found also that the energy consumption of a mobile node increases with the distance between the two communicated nodes. For that, it will be better that the nodes communicate through multi-hop network.

All the above factors affect the performance of a mobile ad hoc network in different ways, but they are all having a great influence on the communication energy consumption. Typically, communicating over wireless medium consumes more battery power than CPU processing [6]. To increase the lifetime of an ad hoc node, it is important to reduce the communication energy cost. On the other hand, energy efficiency does not only depend on the amount of the avoidable extra data, but also on the total duration of the connection [6], meaning that longer connections consume more battery power because of the idle energy consumed and sometimes because of the retransmission action. In other words, if we can minimize the communication time by using robust recovery algorithms to recover from loss error, we may save a lot of energy consumed. M. Zorzi and R.R. Rao [3] found that the efficient usage of energy is achieved when a scheme stops transmitting when the channel conditions become adverse and resumes transmission when they improve. In fact, this is exactly what the window adaptation algorithm of TCP does. That is even though that algorithm was designed for a wire-line environment where energy is usually not an issue; its way of handling congestion turns out to be very efficient in guaranteeing reduced energy consumption as well. In [11] M. Zorzi and R.R. Rao found that as an interesting fact which shads a new light on the effectiveness of TCP as a transport protocol for mobile wireless environments. Also, It has been showed in [17][18][19] that, in the presence of burst packet errors (where many packets are lost at once), backing off transmissions may in fact be the right thing to do, at least as long as error bursts are long relative to the propagation delay of the connection. This is explained by noting that in this case long error bursts are persistent conditions (as is congestion) [11]. In fact, the throughput performance of TCP has been shown to be enhanced by the presence of

bursty errors, as opposed to random errors [17][18][19]. In the following section, we will discuss in more details the different versions of TCP and their impact on the energy consumption of the mobile nodes.

3.5 Conclusion

Energy consumption in a network is not related to only one layer or only one operation, it is a large aspect that contains many factors. It has been found that the energy consumption is directly proportional to the duration of the communication session (connection time). Hence, in our study here, we will use the average connection time as an indication of the energy consumed in the network. In the mean time, we will study the effect of the ad hoc routing protocols on the TCP variants, in order to verify the interaction between the network and transport layers within ad hoc network environment.

CHAPTER IV - Energy Consumption in TCP Variants

4.1 Introduction

When deploying TCP in mobile ad hoc networks, it has been found that TCP is incapable of differentiating between the packet loss due to congested network and that due to channel errors. As known, TCP was initially designed for wired networks, meaning that the major cause of packet loss is the congested routers at the network, while in mobile ad hoc networks; the cause is much more often transmission link errors. This inability of TCP to recognize the main cause of losses in the network, leads to some aggressive actions taken by TCP error recovery algorithms that may waste the limited power resources of the mobile nodes (as will be explained in the following sections). While the receive processes are the same for all of the TCP versions considered, their transmit processes are different in the way the loss recovery phase is implemented. Thus, the energy efficiency of error control strategies cannot be studied without taking into account the associated mechanisms for loss recovery [11].

4.2 TCP Variants

4.2.1 Baseline TCP (Old Tahoe)

Traditionally, the TCP receiver only acknowledges the highest segment it has successfully received in order by an accumulative acknowledgement, and if a segment is lost, there is no way to indicate it to the sender. In the early versions of TCP the only way to recover from a segment loss was to wait for *retransmission timeout* (RTO) to expire. The length of the RTO is determined from the measured *round-trip time* (RTT) and the variance of the recent RTT measurements [1]. The receiver cannot acknowledge new data in case of segment loss and hence the sender is not allowed to send new data. Therefore, before the retransmission timeout expires, there is usually an idle period during which the sender does not transmit any data, possibly causing the communication path to be under-utilized. Also, as mentioned earlier, prolonged communication sessions lead to more power consumption.

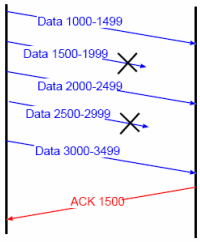


Figure 4.1 Accumulative Acknowledgement used by traditional TCP

As can be seen, when a packet lost the receiver cannot acknowledge new data, then the sender must wait for the retransmit timer to timeout before sending the lost packet, and then the sender will retransmit all the following packets even if they were already sent before.

Traditional TCP consumes a lot of energy at high loss rates, since it will wait for RTO timer to expire each time before retransmitting the lost packet. Also, the energy consumed increases with an increase in RTO. Thus, reducing the RTO value would be a solution to minimize the energy consumption at the sender.

4.2.2 TCP Tahoe

The TCP Tahoe implementation added a number of new algorithms and refinements to earlier implementations. The new algorithms include *Slow-Start* (see Annexe A), *Congestion Avoidance*, and *Fast Retransmit* [7] [10]. The refinements include a modification to the round-trip time estimator used to set retransmission timeout values [7] [1]. The goal of *slow-start* and *congestion avoidance* is to keep the congestion window size around optimal size as much as possible. *Slow-start* increases the window size rapidly to reach maximum safety transfer rate (half of the transfer rate that caused packet loss) as fast as possible and *congestion avoidance* increases the window size slowly to avoid packet losses as long as possible.

The transition from *slow-start* to *congestion avoidance* is done according to the "*ssthresh*" variable traced by TCP:

- If *CWND* < *ssthresh*, then do *slow-start*.
- If *CWND* > *ssthresh*, then do *congestion avoidance*.

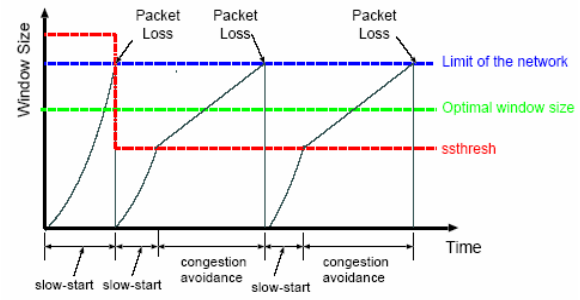


Figure 4.2 CWND Variation of TCP Tahoe

The above graph shows the *CWND* variation of TCP Tahoe at both *slow-start* and *congestion avoidance* phases. And also, it shows how the congestion window adapts itself according to the network conditions. When there is a packet loss that means the *CWND* had to be minimized to overcome the congestion problem in the network.

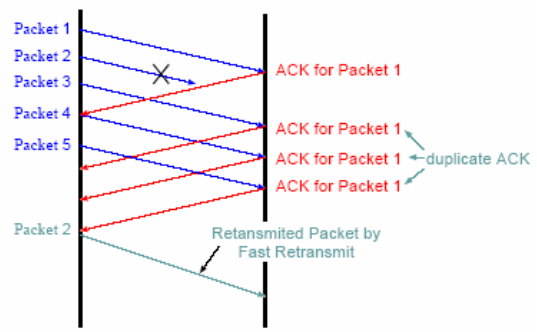


Figure 4.3 Fast Retransmit Algorithm in TCP Tahoe

The sender recognizes that there is a packet loss upon the receiving of three duplicate ACKs from the receiver side. Then, it triggers the fast retransmit algorithm to resend the lost packet without waiting for the RTO to go off. This action will insure the fast retransmission of lost packets, which results in improving the throughput and conserving the consumed energy.

4.2.3 TCP Reno

The congestion control mechanism of TCP Reno, the most popular TCP implementation, retained the enhancements incorporated into TCP Tahoe, but modified the *Fast Retransmit* operation to include *Fast Recovery* [12]. The *Slow-Start* and the *Congestion Avoidance* algorithms are used by a TCP Reno sender to control the amount of data injected into the network while the *Fast Retransmit* and the *Fast Recovery* are used to recover from packet losses without the need for *retransmission timeouts* (RTOs) [2]. The main difference between TCP Tahoe and TCP Reno resides in the congestion estimation part. In TCP Tahoe each packet loss is considered as a serious congestion problem, resulting in setting the *CWND* size to minimum value after each packet loss. While in the Reno version, TCP can differentiate between a two cases:

- If packet loss was found by *Retransmit Timeout*, then the network severs from a serious congestion problem. TCP here sets the window size to minimum value and enters *slow-start* phase.

- If packet loss was found by *Duplicate ACKs*, then the congestion is not severe because of the following:

  1. At least 3 packets could arrive at the receiver after packet loss.
  2. At least 3 packets left the network, so there may be a chance to transmit a packet.

  Then, *CWND* size is set to half the current *CWND* and TCP transits to *congestion avoidance* phase.

The above strategy of data loss recovery in TCP Reno is shown to perform better than TCP Tahoe, when single packet losses occur in one segment (random error); because it is not obliged to wait for the RTO to retransmit the lost packet. However, it can suffer performance problems when multiple packets are lost in one data segment (burst error); since the CWND will be significantly reduced in this case and the return to its original size occurs only after a considerable delay [2]. For example, with two packets lost, the CWND will be reduced twice, while for more than two packets lost the TCP Reno will recognize the loss after RTO expires.

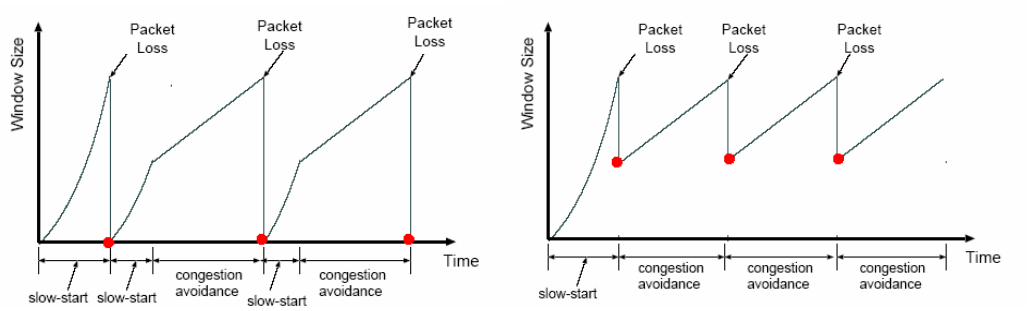


Figure 4.4 CWND variation of TCP Tahoe and TCP Reno

As can be noticed from the above draw, TCP Reno recovers more quickly than TCP Tahoe, since it will not shrink the CWND to minimum each time it will encounter a packet loss as the case in TCP Tahoe. It is important to note that this algorithm of TCP Reno will enhance the network performance and in the same time will decrease the energy consumed at the nodes.

4.2.4 TCP New- Reno

The New-Reno TCP includes a small change to the Reno algorithm at the sender, that eliminates Reno's wait for a retransmit timer when multiple packets are lost from a window (burst error loss) [13] [14].

The change concerns the sender's behavior during *Fast Recovery* when a *partial* ACK is received that acknowledges some but not all of the packets that were outstanding at the start

of the *Fast Recovery* period. In TCP Reno, *partial* ACKs take TCP out of *Fast Recovery* by deflating the usable window back to the size of the congestion window. In TCP New-Reno, *partial* ACKs don't take TCP out of *Fast Recovery*. Instead, *partial* ACKs received during *Fast Recovery* are treated 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, New-Reno can recover without a retransmission timeout, retransmitting one lost packet per round-trip time until all of the lost packets from the window have been retransmitted. TCP New-Reno remains in *Fast Recovery* until all of the data outstanding when *Fast Recovery* was initiated has been acknowledged [2]. The TCP New-Reno version can cover from multiple losses, and is therefore more suited than TCP Reno to the mobile wireless environment, where packet losses may occur in bursts. A major drawback of TCP New-Reno is that the sender retransmits only one packet per RTT. When several losses occur, the TCP New-Reno usually recovers only after a considerable delay [22].

4.2.5 TCP SACK

Traditional implementations of TCP use an acknowledgement number field that contains a cumulative acknowledgement, indicating that the TCP receiver has received all of the data up to the indicated byte. A selective acknowledgement option allows receivers to additionally report non-sequential data they have received. The SACK option is used in an ACK packet to indicate which packets were received precisely. When coupled with a selective retransmission policy implemented in TCP senders, considerable savings can be achieved. Adding SACK to TCP does not change the basic underlying congestion control algorithms. The TCP SACK implementation preserves the properties of TCP Tahoe and TCP Reno of being robust in the presence of out-of-order packets, and uses retransmit timeouts as the recovery method of last resort. The main difference between the TCP SACK implementation and the TCP Reno implementation is the behavior when multiple packets are dropped from one window of data [2]. SACK option is a notification of which segments were received correctly. When a TCP sender receives an ACK that contains a SACK option, it uses this information to decide which packets should be retransmitted [22]. The sender maintains a list of segments deemed to be missing (based on all the SACKs received) and sends new or retransmitted data when the estimated number of packets in the path is less than the congestion window. When a retransmitted packet is itself dropped, the SACK implementation detects the drop with a

retransmission timeout, retransmitting the dropped packet and then *slow-starting*. SACK exits *Fast Recovery* under the same conditions as New-Reno.

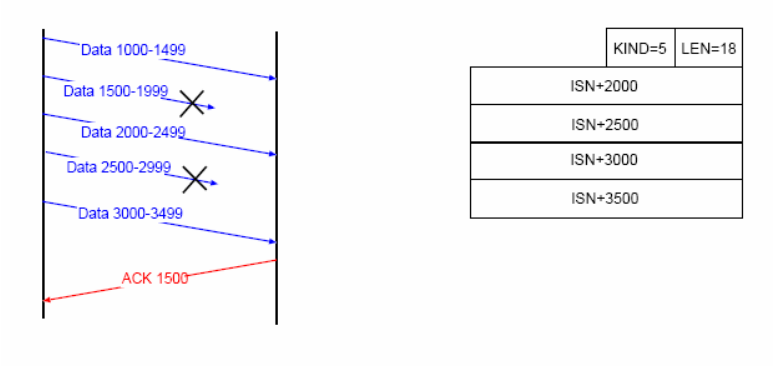


Fig 4.6 Example of the SACK option

The SACK option of TCP will acknowledge the packets that correctly received and the sender will deduce the lost packets from the received SACK coming from the receiver side. In the previous Figure, as an example, the receiver acknowledges that the packets from 2000 to 2500 and those from 3000 to 3500 were received intact, and the sender then will recognize, upon the receiving of that ACK, that the packets between those, which were acked, are lost.

4.2.6 TCP WestwoodNR

TCP WestwoodNR is a sender-side modification of the TCP congestion window algorithm that improves upon the performance of TCP New-Reno in wired as well as wireless networks (in fact, there are two variants of TCP-Westwood, one is based on TCP Reno, and the other is based on TCP New-Reno which we are going to study in this work). The improvement is most significant in wireless networks with lossy links, since TCP WestwoodNR relies on end-to-end bandwidth estimation to discriminate the cause of packet loss (congestion or wireless channel effect), which is a major problem in TCP New-Reno [23].

[When $BWE*RTT_{min}$ (pipe size) << CWND, it is more likely that packet losses are due to congestion. This is because the connection is using a CWND value much higher than its share of pipe size, thus congestion is likely. On the other hand, when $BWE*RTT_{min}$ > CWND, it indicates that packet losses are due to link-error]

The key idea of TCP WestwoodNR is to exploit TCP acknowledgement packets to derive rather sophisticated measurements as follows:

1. The source performs an end-to end estimate of the bandwidth available along a TCP connection by measuring and averaging the rate of returning ACKs.
2. After a congestion episode (i.e. the source receives three duplicate ACKs or a timeout) the source uses the measured bandwidth to properly set the congestion window and the *slow-start* threshold, starting a procedure that is called *faster recovery*.

By backing off to *cwnd* and *ssthresh* values that are based on the estimated available bandwidth (rather than simply halving the current values as Reno does), TCP WestwoodNR avoids overly conservative reductions of *cwnd* and *ssthresh*; and thus it ensures a *faster Recovery,* resulting in achieving higher throughput.

4.2.7 TCP Vegas

Vegas extends Reno's retransmission mechanisms as follows. First, Vegas reads and records the system clock each time a segment is sent. When an ACK arrives, Vegas reads the clock again and does the RTT calculation using this time and the timestamp recorded for the relevant segment. Vegas then uses this more accurate RTT estimate to decide to retransmit in the following two situations: [see 39]

- When a duplicate ACK is received, Vegas checks to see if the difference between the current time and the timestamp recorded for the relevant segment is greater than the timeout value. If it is, then Vegas retransmits the segment without having to wait for *n* (3) duplicate ACKs. In many cases, losses are either so great or the window so small that the sender will never receive three duplicate ACKs, and therefore, Reno would have to rely on the timeout mentioned.
- When a non-duplicate ACK is received, if it is the first or second one after a retransmission, Vegas again checks to see if the time interval since the segment was sent is larger than the timeout value. If it is, then Vegas retransmits the segment. This will catch any other segment that may have been lost previous to the retransmission without having to wait for a duplicate ACK.

In other words, Vegas treats the receipt of certain ACKs as a trigger to check if a timeout should happen. It still contains Reno's coarse-grained timeout code in case these mechanisms fail to recognize a lost segment. Notice that the congestion window should only be reduced due to losses that happened at the current sending rate, and not due to losses that happened at an earlier, higher rate. In Reno, it is possible to decrease the congestion window more than once for losses that occurred during one RTT interval. In contrast, Vegas only decreases the

congestion window if the retransmitted segment was previously sent *after* the last decrease. Any losses that happened before the last window decrease do not imply that the network is congested for the *current* congestion window size, and therefore, do not imply that it should be decreased again. This change is needed because Vegas detects losses much sooner than Reno [39].

First, Vegas sets *BaseRTT* to the minimum of all measured round trip times; it is commonly the RTT of the first segment sent by the connection, before the router queues increase due to traffic generated by this connection. If we assume that we are not overflowing the connection, then the expected throughput is given by:

$$Expected = WindowSize / BaseRTT$$

where *WindowSize* is the size of the current congestion window, which we assume for the purpose of this discussion, to be equal to the number of bytes in transit.

Second, Vegas calculates the current *Actual* sending rate. This is done by recording the sending time for a distinguished segment, recording how many bytes are transmitted between the time that segment is sent and its acknowledgement is received, computing the RTT for the distinguished segment when its acknowledgement arrives, and dividing the number of bytes transmitted by the sample RTT. This calculation is done once per round-trip time.

Third, Vegas compares *Actual* to *Expected*, and adjusts the window accordingly. Let *Diff*= *Expected* - *Actual*. Note that *Diff* is positive or zero by definition, since *Actual* > Expected implies that we need to change *BaseRTT* to the latest sampled RTT. Also define two thresholds, $\alpha < \beta$, roughly corresponding to having too little and too much extra data in the network, respectively. When *Diff* $< \alpha$, Vegas increases the congestion window linearly during the next RTT, and when *Diff* $> \beta$, Vegas decreases the congestion window linearly during the next RTT. Vegas leaves the congestion window unchanged when $\alpha <$ *Diff* $< \beta$.

Intuitively, the farther away the actual throughput gets from the expected throughput, the more congestion there is in the network, which implies that the sending rate should be reduced. The β threshold triggers this decrease. On the other hand, when the actual throughput rate gets too close to the expected throughput, the connection is in danger of not utilizing the available bandwidth. The α threshold triggers this increase. The overall goal is to keep between α and β extra bytes in the network.

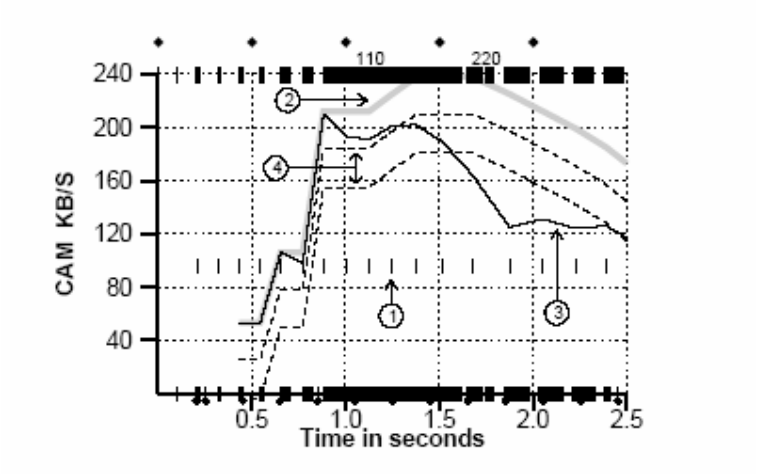


Figure 4.7 Congestion detection and avoidance in Vegas [39].

1. The small vertical line—once per RTT—shows the times when Vegas makes a congestion control decision; i.e., computes *Actual* and adjusts the window accordingly.
2. The gray line shows the *Expected* throughput. This is the throughput we should get if all the bytes in transit are able to get through the connection in one *BaseRTT*.
3. The solid line shows the *Actual* sending rate. We calculate it from the number of bytes we sent in the last RTT.
4. The dashed lines are the thresholds used to control the size of the congestion window. The top line corresponds to the α threshold and the bottom line corresponds to the β threshold.

4.3 Comparative Study of TCP Variants

4.3.1 Simulation Scenarios

Our simulations are done using the network simulator version 2 (NS-2) [40]. The simulations consist of a network of 20 nodes confined in a (670 x 670) m² area. 14 TCP connections were established (ftp traffic used with a packet size of 512 bytes). Simulation time is 400 seconds. The initial battery capacity of each node is 10 joules. This initial energy is reduced progressively by data transmission, reception, retransmission, and forwarding. We consider the simple case where the transmission and reception of a packet consumes a fixed amount of energy from the node's battery. When this initial energy reaches zero joules, the corresponding node cannot take part anymore in the communication, as is regarded as dead. Each node has a radio propagation range of 250 meters. At these simulations, we have

studied the TCP using different loss model scenarios with several values of BER (5%, 10%, and 15%) and a lost link (LL) scenario.

In this work, we are going to study two of the performance parameters of TCP variants: one of them is the energy consumed in transmission, reception, forwarding and retransmission of packets. The other one is the average connection time. Note that, it was proved in the literature [6] that this time is proportional to the sum of the energy consumed at listening the radio channel plus that consumed at the execution of the different algorithms used in each TCP variant at the CPU unit.

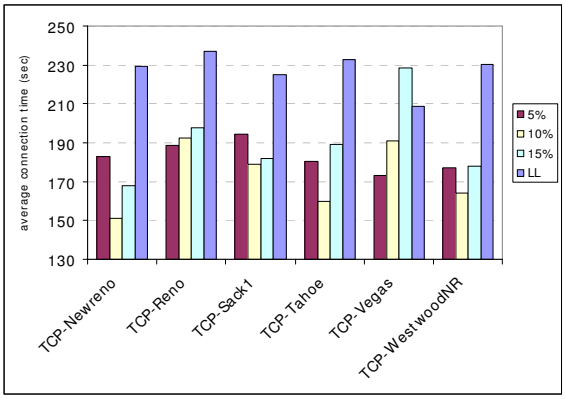


Figure 4.8. Average connection time of TCP sessions

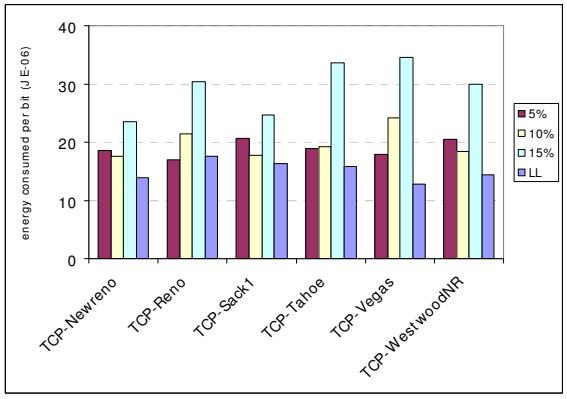


Figure 4.9. Energy Consumed per Received Bit

4.3.2 Energy Consumption of TCP Tahoe

From Figure 4.9, we can conclude that our results confirm the results obtained in [11] in which that TCP Tahoe performs better in the case of burst errors (lost link) than in the case of random BER (as can be seen in the Figure). TCP Tahoe saves energy consumption per received bit because it backs off in the presence of burst loss, which confirms the fact that, at burst error case or when there is a lost link in the network, it is better to stop data transmission until a new route is found. In [11] the authors leave the important issue of the energy efficiency tradeoffs involved when backing off increases delays, and hence the overall connection time. Figure 4.8 can explain the effect of "backing off" algorithm of TCP Tahoe on the average connection time and hence on the total energy consumption of TCP. Although that backing off saves energy consumed per received bit in the burst error case, we have found that it introduces an extra delay in the network. And as discussed before, the longer the connection time, the greater the total energy consumed due to the idle energy consumption at the node. At idle times, although that the node does not send data, it will listen to the radio waves in order to receive the acknowledgement, also there is a time

consumption at the CPU unit in which it executes the TCP algorithms used (*fast retransmit*, *slow-start*, and *congestion avoidance* algorithms). At high BER, TCP Tahoe consumes a great amount of energy due to high number of unnecessary retransmissions and the fact that after each error recovery TCP Tahoe will enter the *slow-start* phase, which means slowing down its transmission rate.

4.3.3 Energy Consumption of TCP Reno

The simulation results in Figure 4.9 proved that the energy consumed by TCP Reno will be less than that of the Tahoe version, in the case of random loss, due to the *fast recovery* action taken, which does not exist in the TCP Tahoe. However, in the case of bursty packet loss (lost link), TCP Tahoe may have lower energy consumption per received bit, since it backs off in front of the burst errors, which may increase the chance of successful retransmission after that. For example, if the burst packet loss is due to a bad connection or a link failure, backing off for a while, will help avoiding the unnecessary retransmissions. On the other hand, Figure 4.8 shows that TCP Reno has a long average connection time compared with the other TCP variants especially at high BER due to the fact that TCP Reno is unable to recover from more than one lost packet at a time. When there is a high BER, TCP Reno decreases its transmission rate by half each time there is a lost packet, then after two trials of loss recovery, TCP Reno reaches almost the same transmission rate as in TCP Tahoe. After three trials of recovery, TCP Reno has to wait to RTO expiration that leads to backing off and entering *slow-start* phase (exactly as in TCP Tahoe). The above process leads to more time consumption in the first two trials of recovery, while that TCP Tahoe is backing off and goes through *slow-start* directly from the first packet loss. As TCP Reno has a high average connection time compared with TCP Tahoe, it means that TCP Reno consumes more energy than TCP Tahoe. Indeed, as mentioned before, the average connection time is directly proportional to the energy consumed at the CPU in order to execute the TCP variant algorithms plus that consumed at listening to the radio channel. Hence, we can conclude that there is an extra energy consumed in order to execute the *fast recovery* algorithm in the CPU. This led us to conclude that TCP Reno will probably consume more total energy than TCP Tahoe in almost most of the cases.

4.3.4 Energy Consumption of TCP New- Reno

In this variant of TCP, there will be a noticeable savings in the energy consumption per received bit in both random and bursty packet losses due to the *partial ACKs* used, since it will not wait for RTO timer in order to retransmit the lost packet. For bursty loss errors, TCP Tahoe performs better than TCP Reno (as explained earlier), and TCP New-Reno performs slightly well than Tahoe [Figure 4.9], since TCP New-Reno is not obliged to wait before retransmitting the lost data and in the mean time its congestion window will increase faster than that of TCP Tahoe. Figure 4.8 shows that TCP-New-Reno has less average connection time compared with both TCP Tahoe and TCP Reno, at burst error case and at high BER, due to the *fast recovery* algorithm which is not available in TCP Tahoe and partial ACKs that does not exist within TCP Reno.

We can also conclude from Figures 4.9 and 4.8 that TCP New-Reno has always the best performances compared to all other TCP variants in terms of energy consumption per received bit.

4.3.5 Energy Consumption of TCP SACK

Figure 4.9 shows although that TCP SACK does not consume a lot of energy in ad hoc networks; TCP New-Reno slightly outperforms it in term of energy consumed per received bit at different BER values, and even at lost link case. In fact, when comparing the general performances (goodput) of TCP SACK and TCP New-Reno over wireless channels, we can note that TCP SACK outperforms TCP New-Reno in most of the cases. This is mainly due to the *Selective Acknowledgements* feature that allows TCP SACK to terminate the retransmission of lost data more quickly than TCP New-Reno (which does not know which of the unacked segments are missing at the receiver). However, in terms of energy consumption, this gain is neutralized. We think that this is due to the overhead that is introduced by the SACK option. Indeed, SACK packets[1] can in certain cases reach the double of normal TCP ACK packet size[2]. This will in turn lead to more energy consumption per sent SACK.

Figure 4.8 is showing two different results:

[1] SACK packet size = IP Header + TCP ACK Header + SACK option = 20 bytes + 20 bytes + 40 bytes = 80 bytes. 40 bytes is the maximum size of a TCP Header option. SACK can use this entire size to transmit the *Selective Acknowledgement*. This size depends on the number of segments to be acknowledged.
[2] Normal TCP ACK packet size = IP Header + TCP ACK Header = 20 bytes + 20 bytes = 40 bytes.

- The first is that TCP SACK has a shorter average connection time in case of burst loss (lost link) as it retransmits only the lost packets (as explained earlier). The following packets, which may use the recovered route (by the routing protocol), are not retransmitted. In the mean time, TCP New-Reno has to retransmit the entire segment even the packets not loosed. Furthermore, TCP New-Reno can not retransmit more than one lost packet per RTT, which means that at burst loss case (lost link), we must wait for a time equal to the number of packets considered as lost multiplied by the RTT value. Hence, the large the number of lost packets, the longer we must wait to recover. This is not the case of TCP SACK that retransmits all the lost packets without waiting for RTT.
- The second result which can be viewed from Figure 4.8 is that at high BER TCP New-Reno has a shorter average connection time than that of TCP SACK. This is due to the fact that at high BER, we may have a retransmitted packet that is dropped again, the SACK implementation detects the drop with a *retransmission timeout*, retransmitting the dropped packet and then *slow-starting*, which is not the case with TCP New-Reno. Indeed, TCP New-Reno does not enter the *slow-start* phase, but it decreases the transmission rate by half the current value.

However, note that the energy consumption (per time unit) due to the operation of the algorithms of TCP SACK (CPU units) is high compared to TCP New-Reno. Then, even if, TCP SACK has a slightly low average connection time in the burst error case (link lost), the energy consumed by TCP SACK may be higher then the one consumed by TCP New-Reno.

4.3.6 Energy Consumption of TCP WestwoodNR

Figure 4.8 showed that TCP WestwoodNR has the same average connection time as that of TCP New-Reno in the case of burst error (lost link). Indeed, at lost link case, a TCP session will experience burst losses then both TCP variants will have to wait until the RTO expires before retransmitting. As both of them wait for an RTO, this can gives the routing protocol enough time to recover the route between source and destination. This will in turn lead to approximately the same average connection time for both variants. Figure 4.8 proves that at low BER (5%), TCP WestwoodNR will have shorter average connection time than TCP New-Reno, because of its ability to adjust the transmission rate according to the bandwidth estimated in the network instead of blindly halving it as with TCP New-Reno. Also, Figure 4.8 demonstrates that TCP New-Reno outperforms TCP WestwoodNR at high BER (10% and 15%) in terms of average connection time, because at high BER there is a loss at the received

ACKs leading to misbehavior of the bandwidth estimation algorithm of TCP WestwoodNR (it is also the case in Figure 4.9 with the energy consumed per received bit).

If we look at Figure 4.9, we can see the same situation as when looking to Figure 4.8. TCP WestwoodNR and TCP New-Reno consume approximately the same energy in the cases of link loss (for the same reasons as for the average connection time) and for low and medium BER. For low and medium BER (5% and 10%), even if TCP sessions do not take the same average connection time when using TCP New-Reno or TCP WestwoodNR, the proportion between received bits (by the destination) and the total sent bits including lost packet (by the source) remain almost the same. This is due to the operations of both TCP variants that can introduce, depending on the situation (BER), more or less delay when retransmitting lost packets. More precisely, we think that the size of the congestion window is not well dimensioned by both TCP variants depending on the BER. Hence, sometimes TCP WestwoodNR has the best transmission rate and some other times it is TCP New-Reno.

4.3.7 Energy Consumption of TCP Vegas

When dealing with low BER (5%), TCP Vegas will be the best of all TCP variants in terms of energy consumed per received bit and average connection time [as can be seen from Figures 4.8 and 4.9]. This is due to the behavior of TCP Vegas that, on the first duplicate ACK received checks for the RTT value and compares it with the RTO value leading to faster recovery than the other TCP variants that have to stay until the reception of the third duplicate ACK. At high BER (10% and 15%), TCP Vegas has always the worst average connection time and energy consumed per received bit, because of its dependence on the RTT measured values of the received ACKs. Hence, at high BER, we will have a high loss in received ACKs that in turn will force TCP Vegas not to have a good behavior. In addition, it is shown that TCP Vegas has low average connection time in case of burst error (lost link) due to its ability to deduce a good estimation for the transmission rate compared to TCP New-Reno (that simply halves the congestion window size) or TCP WestwoodNR (that waits for the third duplicate ACK arrival before entering the retransmission process). This behavior also leads to less energy consumption as can be verified from Figure 4.9.

4.3.8 Summary

To summarize simulation results obtained in this chapter, at high BER (15%) TCP New-Reno outperforms the other TCP variants followed by TCP SACK. In low and medium BER (5%

and 10%), TCP New-Reno has also a good energy consumption per received bit ratio. In addition, TCP Vegas and TCP New Reno have the best (the least) energy consumption when there is a lost link in the network.

When comparing the average connection time, we found that at medium and high BER (10% and 15%) TCP New-Reno outperforms the other TCP variants followed by TCP WestwoodNR. In addition, TCP Vegas and TCP SACK, followed by TCP New-Reno, have the best (the least) average connection time when there is a lost link in the network. On the other hand, when we have a low BER (5%), the best TCP variants, in term of average connection time, are Vegas and WestwoodNR. This is due to their ability, at this BER level, to adjust well the congestion window size.

Our results show that in almost all studied situations, TCP New-Reno is the one having the best performances in terms of energy efficiency in a static ad hoc network.

4.4 Conclusion

It was proved that the congestion control algorithm in TCP variants has an important effect on the energy consumption in an ad hoc network. Even that TCP was originally designed for ordinary wired networks, it has been proved that it will fit well in conserving the energy in ad hoc networks especially in front of burst error cases which is, in most of the time, a consequence of a lost link in the ad hoc network. Also, we found that the average connection time of a TCP session can give a good indication of the delay introduced in the network, leading to energy consumption. From our comparative study in this chapter, we conclude that TCP New-Reno is the best among all other TCP variants, because of its ability to handle both random BER and broken link losses efficiently. However, in the above scenarios there is no mobility introduced in the network and then no effect due to mobility handling by ad hoc routing protocols. Now, as we studied the behaviour of TCP variants in static ad hoc networks by varying the type and the importance of losses, we will study (in the next chapter) the effect of other ad hoc network parameters (routing protocols and mobility factor) on TCP performance parameters (energy consumption and average connection time).

CHAPTER V – Energy Consumption of TCP Variants in Mobile Ad Hoc Networks

5.1 Introduction

The energy consumption in ad hoc networks is affected by several factors; among the most important ones, we can find the degree of mobility and the impact of the choice of the routing protocol. In this chapter, we will study the effect of both of them on the energy consumed by TCP variants.

5.2 Simulation Scenarios

Here, we have used the same scenarios used in the last chapter, with four different node velocities (5, 15, and 30 m/s) combined with four different routing protocols (DSDV, AODV, DSR, and OLSR). Our aim is to study the effect of each of them on the energy consumption of each TCP variant. The purpose of our study is to find the most suitable TCP variant in different network environments.

5.3 Routing Protocol Effects on TCP Energy Consumption

To study the effect of the routing protocol on the TCP variants performances, we compare their behavior according to each node velocity value (5, 15, and 30 m/s).

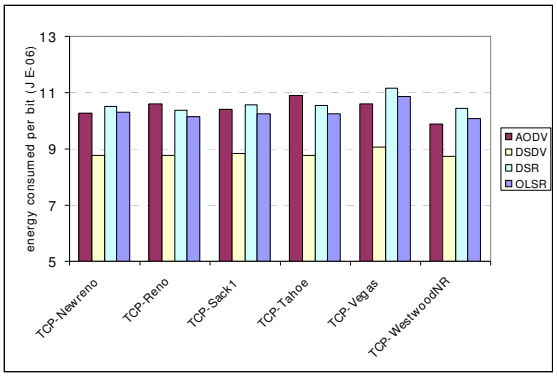


Figure 5.1.a Energy consumed per received bit at 5 m/s speed

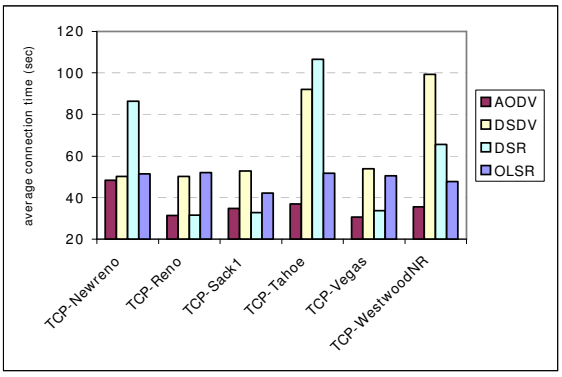


Figure 5.1.b Average connection time of TCP connections at 5 m/s speed

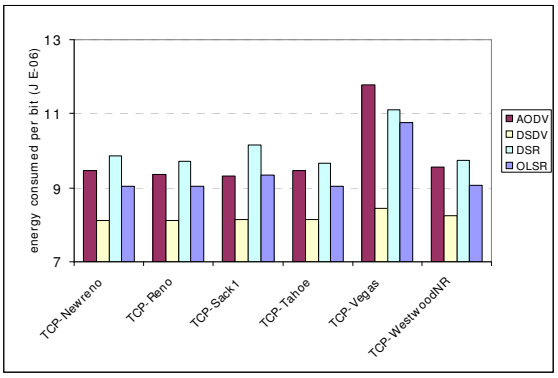


Figure 5.2.a Energy consumed per received bit at 15 m/s speed

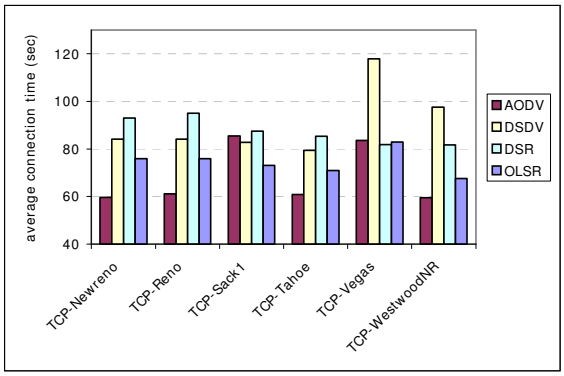


Figure 5.2.b Average connection time of TCP connections at 15 m/s speed

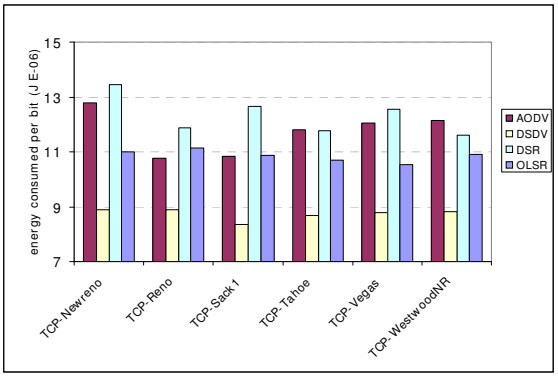


Figure 5.3.a Energy consumed per received bit at 30 m/s speed

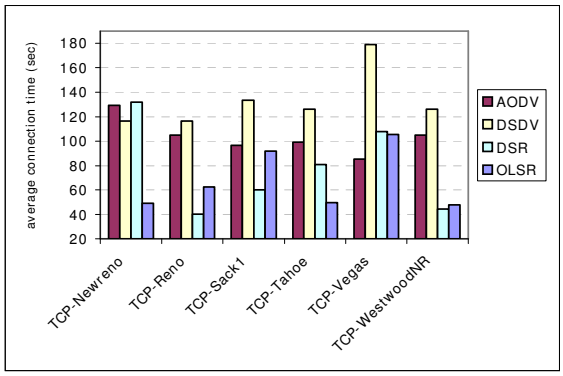


Figure 5.3.b Average connection time of TCP connections at 30 m/s speed

TCP traffic uses two routes: (i) one from the source to the destination and (ii) another in the opposite direction. This means more traffic control in the network when using reactive protocols if the links are asymmetric in the network. In contrast, these routes are provided by default in the topology routing tables in proactive protocols. In addition, when there is a lost link in the network, reactive protocols are obliged to search for new valid route, while in proactive protocols, there is no need to search again as they have redundant routes in their routing protocol tables. That is why proactive protocols have less energy consumed per received bit compared with reactive ones, especially at high mobility rates [Figures5.1.a, 5.2.a, 5.3.a]. We also note that the more the number of nodes in the network, the larger the topology table updates, meaning that the overhead due to topology updates caused by proactive protocols in this case will consume a lot of network resources (time and energy).

In the following, we will study in more details the effect of ad hoc routing protocols on TCP variants in terms of energy consumed per received bit and the average connection time.

5.3.1 Effects of AODV on Energy Consumption of TCP Variants

Figure 5.1.a shows that AODV has high energy consumed per received bit because of its route discovery process overhead. TCP Tahoe consumes more energy per each received bit because it enters the slow-start phase after each time there is a lost link. Let us recall that each time there is a lost link in the network, AODV try to find a new valid route to handle the interrupted communication (i.e., a TCP session in our case). But, as TCP Tahoe will decrease its transmission rate significantly (slow-starting), and as there is a lot of lost links at the network due to node mobility, TCP Tahoe is obliged to decrease its transmission rate many times leading to less data to be sent [Figure 5.1.a]. Hence, the ratio between the overhead induced by AODV control packet and the amount of data received will be the highest leading to an important energy consumption per received bit for TCP Tahoe. This is not the case with TCP Reno and TCP New-Reno. Indeed, these two variants 'only' half their transmission rate instead of entering the slow-start phase. While TCP WestwoodNR has the best energy consumption due to its way to adjust the transmission rate according to the bandwidth estimated in the network. Figure (5.1.b) shows that, AODV has the shortest average connection time compared with the other routing protocols, because at low mobility speed networks, the route discovery process does not happen so frequently. At medium mobility speed (15m /s), all the TCP variants have comparable values of energy consumption per received bit except that of TCP Vegas, which has a higher value [Figure 5.2.a]. This is because TCP Vegas relies on the RTT estimated value to decide its transmission rate. The change in RTT value may lead to decreasing the transmission rate, hence consuming more energy. Figure (5.3.a) shows that at high mobility speed (30m/s) TCP Reno and TCP SACK have the best performance in terms of energy consumed per received bit.

5.3.2 Effects of DSR on Energy Consumption of TCP Variants

Even that DSR is a reactive routing protocol and does not send periodically routing advertisement, but the simulations showed that, it consumes more energy than the other protocols [Figure 5.3.a] because of its routing overhead. DSR packets carry full path information, which means sending a lot of routing information within the sent packet, leading to less user data sent per TCP session despite that it decreases the overhead of intermediate nodes (which route the packet depending on this information). As a result, DSR is not the best choice if we have many data to send.

On the other hand, DSR allows the nodes to keep multiple routes to the same destination in their caches, meaning that when a broken route is found in the network the source node can check its cache for another valid one. If the node can find another route, the route discovery process is not needed, resulting in faster route recovery. This may explain that in the cases of TCP Reno and Tahoe [see Figure 5.1.a], TCP Vegas [see Figure 5.2.a] and TCP WestwoodNR [see Figure 5.3.a] DSR consumes less energy than AODV, which is not the case with other TCP variants here.

In addition, we can see that DSR consumes more time than AODV [Figures 5.2.b, 5.1.b], because of the processing time of packets (due to the heavy routing information overhead). On the other hand, at high mobility rates (30 m/s), DSR may have less average connection time than AODV due to its cache which reduces the time consumed at route discovery process [Figure 5.3.b]. Actually, the long average connection times of TCP New-Reno and TCP Tahoe (see Figure 5.1.b) are due to the behavior of both of them. TCP Tahoe backs off for a long time, and TCP New-Reno can not retransmit more than one lost packet per RTT, which means that the larger the number of lost packets, the longer the time consumed to recover, results in longer connection time. Figures (5.1.a), (5.2.a) and (5.3.a) show that TCP Reno, TCP Tahoe, and TCP WestwoodNR are the best among the other TCP variants when using DSR routing protocol at different mobility speeds. This is because TCP Tahoe backs off when there is a loss in the network, which will conserve the consumed energy. In addition, at low mobility speed (5m/s), TCP Reno is a good choice as there is no high loss in the network (and the DSR buffer may help in this case, with buffering the packets until finding a new route will save the retransmission process at TCP Reno). While in high mobility speed (30m/s), TCP Reno will back off exactly as in TCP Tahoe. For TCP WestwoodNR, the bandwidth estimation algorithm saves the energy consumed by adjusting its transmission rate according to the available bandwidth in the network.

5.3.3 Effects of DSDV on Energy Consumption of TCP Variants

We can see from Figures (5.1.a), (5.2.a) and (5.3.a) that DSDV routing protocol consumes less energy per received bit than the other routing protocols because it does not have to wait for a route discovery process to find a new route (in contrast to AODV and DSR).

Figure (5.1.b) shows that at low mobility (5m/s), TCP Tahoe and TCP WestwoodNR have long average connection time because of backing off algorithm in TCP Tahoe and bandwidth estimation algorithm in TCP WestwoodNR that introduce some delay in the network. While

at high mobility speeds (15m/s, and 30m/s), TCP Vegas has the longest average connection time because of the continuous change in propagation delay which causes TCP Vegas to modify its transmission rate frequently, results in long connection time [Figures (5.2.b) and (5.3.b)]. We find also that all TCP variants average connection time increases with the mobility speed due to the large number of topology changes. DSDV does not have a buffer. Hence, when there is a lost link in the path, DSDV will drop the packet forcing TCP variants to initiate the congestion control algorithm and resulting in large number of retransmissions. We can see from figures (5.1.a), (5.2.a), and (5.3.a) that all TCP variants have comparable energy consumption per received bit at different mobility speeds. That is because DSDV routing protocol drops the packets that could not be routed immediately (lake of buffering feature) which will force almost all the TCP variants to back off in front of packet losses.

5.3.4 Effects of OLSR on Energy Consumption of TCP Variants

The control traffic of AODV and DSR varies depending on the topological changes in the network (caused by mobility and broken links) while it is constant for OLSR.

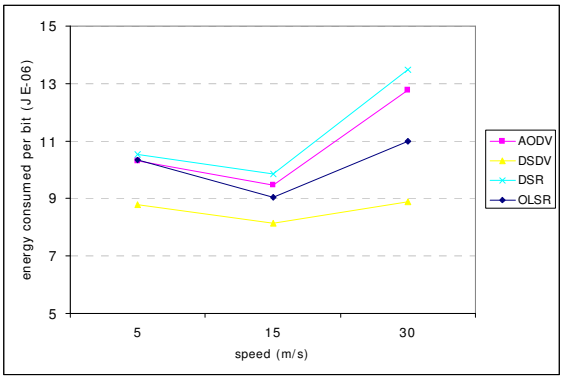
Where DSDV routing protocol uses merely the number of hops to make routing decisions (as a distance victor routing protocol), OLSR performs more complex metric calculations (as a link state protocol) to construct its routing table. As a result, we find that OLSR consumes more energy per received bit than DSDV. In addition, OLSR diffuse its entire routing information table periodically, on the contrary of DSDV, which sends only the routes vectors. As in DSDV, the most TCP variants have almost the same performance in terms of energy consumed per received bit because that OLSR does not buffer the packets that it cannot route for the moment. Hence, forcing TCP variants to back off and then initiate their different retransmission algorithms. All the figures show that TCP Vegas has the worst performance in terms of energy consumed per received bit and average connection time. This is due to its reaction when there is a change at the propagation delay in the network caused by the mobility. TCP Vegas may decrease its transmission rate depending on the measured propagation delay in the network, which leads to more energy consumption. While the other TCP variants will have, less energy consumed per received bit than TCP Vegas because they do not rely on the propagation delay of the network to adjust their transmission rate.

5.3.5 Summary

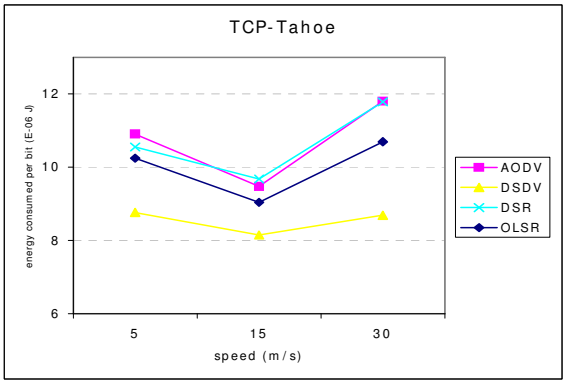
The simulation results prove that, reactive protocols (AODV and DSR) consume more energy per received bit than the proactive ones because of their routing overhead (route discovery, route maintenance). In addition, TCP WestwoodNR is the best choice with AODV at low mobility speed, while TCP Reno and TCP SACK have the best performance in terms of energy consumed per received bit at high mobility speed when using AODV as a routing protocol in the network. With DSR, TCP Reno, TCP Tahoe and TCP WestwoodNR are the best choice among the other TCP variants at different mobility speeds. On the other hand, we find that all TCP variants have almost about the same energy consumed per each received bit when using DSDV. In OLSR, TCP Vegas has the worst energy consumption per received bit due to its dependence on the propagation delay measurement of the network.

5.4 Mobility Effects on TCP Energy Consumption

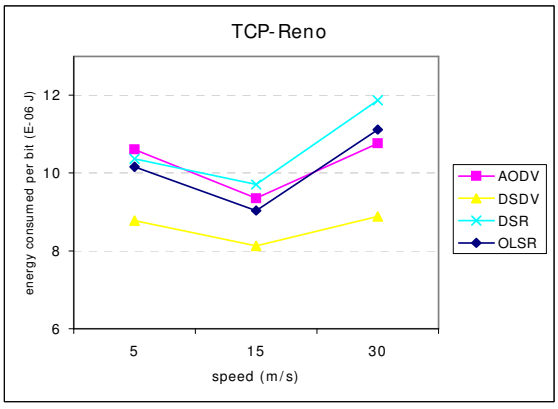
In this section, we are interested to find the effect of different velocities on each TCP variant.



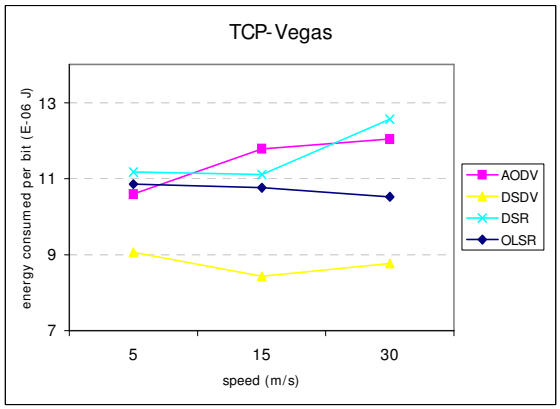
(a)



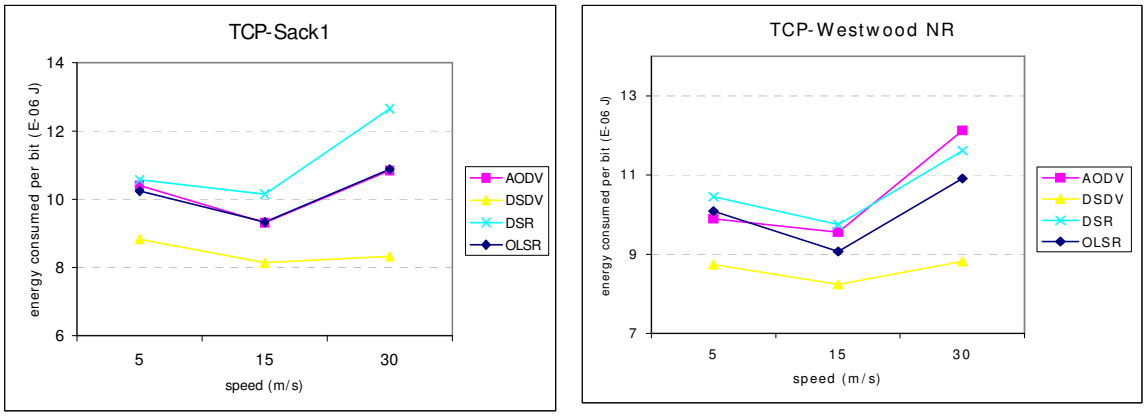
(b)



(c)



(d)



(e) (f)

Figure 5.4 The energy consumed per received bit in: (a) TCP New- Reno (b) TCP Tahoe (c) TCP Reno (d) TCP Vegas (e) TCP SACK (f) TCP WestwoodNR

5.4.1 Mobility Effects on Energy Consumption

5.4.1.1 TCP Tahoe Energy Consumption

The above Figure 5.4 (b) shows that, DSDV is the best choice to implement in ad hoc networks with TCP Tahoe as it has the lowest energy consumption per received bit compared with the other routing protocols. We can also see that OLSR outperforms both AODV and DSR because of its constant periodic updates. The reason that DSDV outperforms OLSR is that DSDV can send an incremental update when there is a topology change in the network without the need of diffusing its entire routing table. At high mobility speed (30m/s) reactive protocols (AODV and DSR) have the same energy consumed per received bit due to the high number of topology changes which leads to many route discovery processes in the network.

5.4.1.2 TCP New- Reno Energy Consumption

Figure 5.4 (a) shows that, DSDV routing protocol is performing well with TCP New-Reno at different mobility speeds due to its incremental updates. On the other hand, at low mobility rate, AODV, DSR and OLSR have the same energy consumed per received bit because at low mobility speed the OLSR routing updates are comparable with the route discovery process in reactive protocols as there are no many topology changes in the network. We see some difference between the three protocols as the mobility rate increases, and OLSR is the favourable one among them with that increase of speed. At high mobility speed (30m/s) the overhead of route discovery process of reactive protocols augments.

5.4.1.3 TCP Reno Energy Consumption

Figure 5.4 (c), shown above, proves that DSDV is the best choice as an ad hoc routing protocol when using TCP Reno. We can see also from the same Figure that at low mobility rates, there is a little nuance between the other three protocols. However, at higher mobility rates, DSR starts to consume more energy than the others because that DSR packet carries full path information [as explained in section 5.3.2].

5.4.1.4 TCP SACK Energy Consumption

Figure 5.4 (e) shows that both AODV and OLSR routing protocols have exactly the same performance especially at high mobility speeds (15m/s and 30m/s) with TCP SACK. At high network topology changes the overhead of route discovery processes in AODV is high, and in OLSR, when there is no valid route in the routing table, OLSR will drop the packets (OLSR does not have a buffer) which leads to high number of packet loss and then high retransmissions, which means consuming more energy in retransmitting the lost packets. In other words, the energy consumed at route discovery process by AODV at high mobility network will be equal to the energy consumed at retransmitting the lost packets when using OLSR at TCP SACK case. While DSDV still has the best performance and DSR consumes more energy per each received bit due to its routing overhead (adding the entire routing path to each sent packet).

5.4.1.5 TCP Vegas Energy Consumption

As can be seen from Figure 5.4 (d), when implementing TCP Vegas in the network, DSDV is having the best performance in terms of energy consumed per received bit. The three other protocols do not have a big difference at low mobility rate, while OLSR is outperforming AODV and DSR at high mobility rate networks due to its constant routing updates.

5.4.1.6 TCP WestwoodNR Energy Consumption

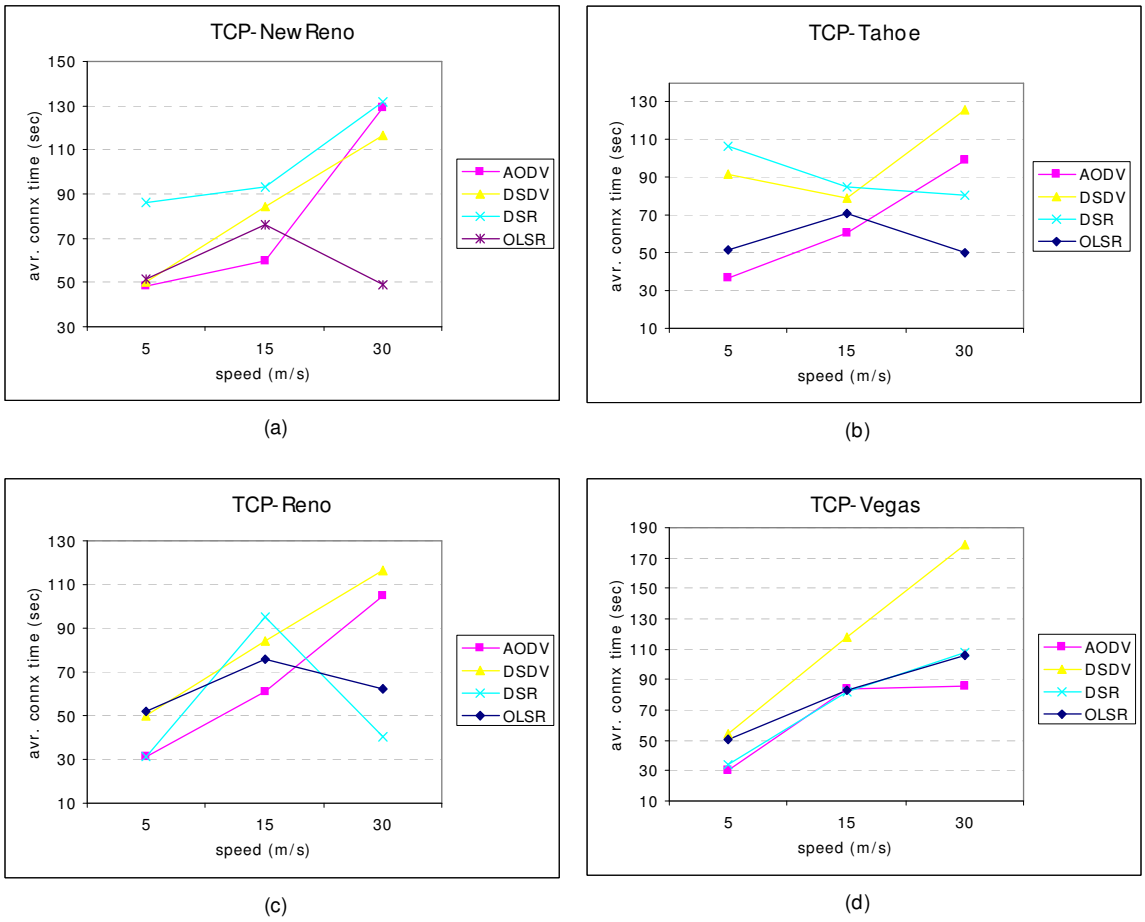
Figure 5.4 (f) shows that DSDV is a favourable routing protocol when implementing TCP WestwoodNR in the ad hoc network. And at high mobility speed (30m/s), AODV has the highest energy consumed per received bit. DSR has less energy consumption than AODV at high mobility speeds due to its cached routes.

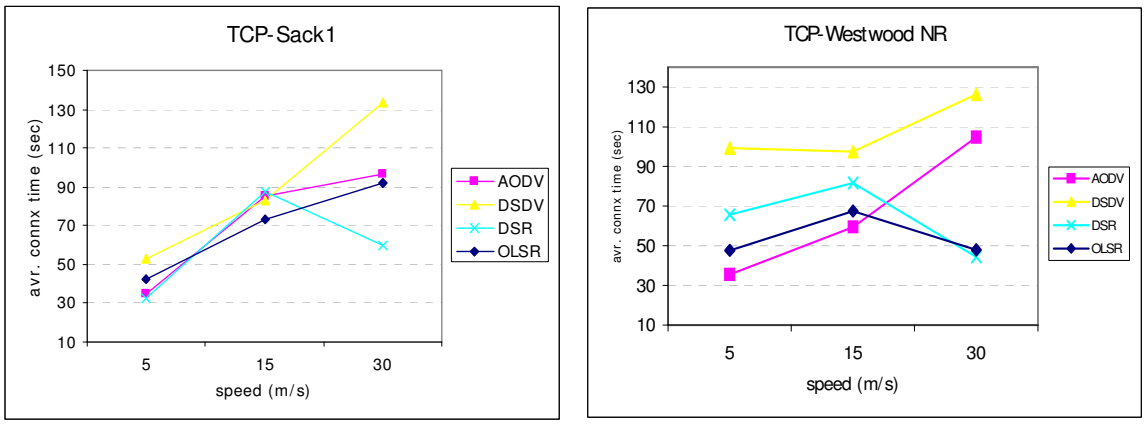
5.4.1.7 Summary

From the above observations, we can conclude that in terms of the energy consumed per received bit, proactive routing protocols outperform the reactive ones in most cases, as there

is no need for initiating the route discovery process in each time there is a lost link in the network. DSDV outperforms OLSR due to its incremental updates. In addition, it was shown that, reactive protocols (AODV and DSR) energy consumption per each received bit increases as the mobility speed increases due to the overhead of the route discovery process. In some cases, such as TCP WestwoodNR, DSR has less energy consumption than AODV due to the routes in its cache.

5.4.2 Mobility Effects on Average Connection Time





(e) (f)

Figure 5.5 The average connection time of: (a) TCP New- Reno (b) TCP Tahoe (c) TCP Reno (d) TCP Vegas (e) TCP SACK (f) TCP WestwoodNR

5.4.2.1 TCP New- Reno Average Connection Time

From Figure 5.5 (a), we can conclude that with TCP New-Reno, the average connection time is almost the same when AODV, DSDV or OLSR are used at low mobility speed due to insignificant network topology changes. DSR consumes more time due to its heavy routing information overhead when compared with AODV. On the other hand, at high mobility speeds, OLSR will have the least average connection time because it searches the shortest route in the network in terms of link delay; leading to less consumed time for sending and receiving data. DSDV has high average connection time because it selects the shortest route in terms of number of hops, which is not necessarily the optimum (the chosen links may have high delay or be congested).

5.4.2.2 TCP Tahoe Average Connection Time

We can see from Figure 5.5 (b) that AODV is preferable with TCP Tahoe at low mobility speed (5m/s) due to its short average connection time because at low mobility speed there are no many topology changes in the network, hence the route discovery process is not frequently initiated which does not introduce high latency in the network. OLSR is the best when we have a high mobility speed (30m/s) for the same raison explained before. Also, we notice that AODV average connection time increases with the mobility speed because of the high number of route discovery process initiated in the network (high mobility rate means a lot of lost links in the network topology). On the other hand, we can see that, DSR average connection time decreases with the increase of mobility speed due to its cached routes.

5.4.2.3 TCP Reno Average Connection Time

Figure 5.5 (c) shows that for TCP Reno, both AODV and DSR (reactive protocols) will have the shortest average connection time at low mobility speed, while that DSR will be the best choice at high mobility speed because DSR may have many valid routes in its cache that will conserve the time used in route discovery process.

5.4.2.4 TCP Vegas Average Connection Time

We can see from Figure 5.5 (d) that reactive protocols will have the shorter average connection time than the proactive ones at low mobility speed with TCP Vegas. At high mobility speed, AODV has the best average connection time (the shortest) because AODV has a buffer to keep the unsent packets until finding a new valid route and as TCP Vegas measures the propagation delay in the network when it receives the acknowledgement. Hence, as long as the delay introduced by the route discovery is less than RTO, TCP Vegas does not have to resend the packet, which means conserving the time of retransmitting packets.

5.4.2.5 TCP SACK Average Connection Time

Figure 5.5 (e) shows that, when using TCP SACK in an ad hoc network, AODV and DSR will have the same performance in terms of the average connection time at low mobility speed, while, DSR will consume less than AODV at high mobility speed due to the use of a cache in DSR.

5.4.2.6 TCP WestwoodNR Average Connection Time

Figure 5.5 (f) shows that, in ad hoc network with TCP-Westwood, AODV has the shorter average connection time, at low mobility speed, compared with the other routing protocols due to its low route discovery overhead.

5.4.2.7 Summary

To summarize, we can say that AODV has a good performance when we have a low mobility speed in the network. In addition, we find that DSR performs well at high mobility speeds. For OLSR, we think that it is the routing protocol, which has the most accepted performance compared to the other ones.

5.5 Conclusion

In this chapter, our simulations showed that, when comparing the energy consumption per received bit of TCP variants, DSDV is the best choice at all mobility speeds. In addition, for the average connection time, each TCP variant is having a different behaviour with the different ad hoc routing protocols. On the other hand, we find that OLSR performs well with the most TCP variants at different mobility speeds.

CHAPTER VI - Conclusion and Perspectives

TCP was mainly designed for wired networks, where the main cause of lost packets is the network congestion. In ad hoc networks, there are multiple reasons for losing packets. Among them the (i) lost links due to node's mobility and (ii) BER of wireless links that can be more or less high depending on the situation. Hence, it has been found that TCP congestion control may have some aggressive actions dealing with packet loss in ad hoc networks. In addition to that, ad hoc networks have limited resources in terms of energy, hence we found important to study the characteristics of TCP energy consumption in such environment. Our work consisted of studying the TCP variants in terms of energy consumption in different possible situations (BER, routing protocols, and mobility).

Simulation results showed that TCP New-Reno is the best choice, in terms of energy efficiency (energy consumption per received bit, average connection time), for a static ad hoc network. This is because partial ACKs improve its reaction in front of lost links and more or less high BER. We have also found that proactive routing protocols, especially OLSR, lead TCP variants to have better behavior in terms of energy efficiency than reactive ones. Indeed, when comparing the average connection times of different scenarios, we found that, at low mobility speed, reactive protocols had shorter average connection time than proactive protocols, while at high mobility speed, the time-overhead of reactive protocols (used for route discovery and route maintenance) might increase the average connection time significantly.

Let us remind that, it have been found that the average connection time of TCP session is a helpful way to estimate the energy consumed by TCP operations (idle-energy/channel-listening and processing at CPU unit). For future works, we propose to find a methodology in order to translate the average connection time into energy units (joules) leading to a more accurate study. In addition, it will be more useful to introduce more accurate error models to better understand the performances (energy and average connection time) of TCP variants. We propose also, to study TCP-variants performances in the presence of efficient routing protocols[3] to find out if they can reduce significantly the amount of energy consumed by TCP in ad hoc networks or not.

[3] Efficient routing protocols aim to conserve the energy of the node's battery.

Annexe A

A.1 *SLOW-START* Algorithm

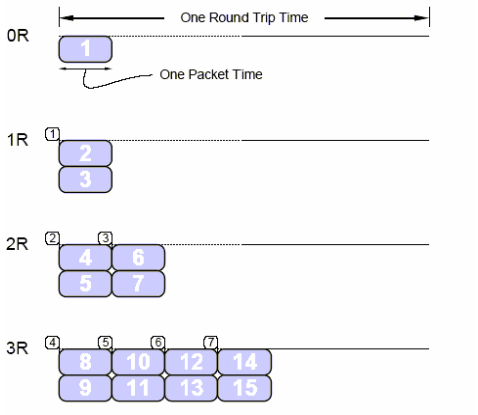


Figure A.1. Slow–Start Algorithm [7]

The horizontal direction is time. The continuous time line has been chopped into one round-trip-time pieces stacked vertically with increasing time going down the page. The grey, numbered boxes are packets. The white numbered boxes are the corresponding ACKs. As each ACK arrives, two packets are generated: one for the ACK (the ACK says a packet has left the system so a new packet is added to take its place) and one because an ACK opens the congestion window by one packet [7].

The *slow-start* works as follows, at the sender [7]:

- Add a *congestion window*, *CWND*, to the per-connection state.
- When starting or restarting after a loss, set *CWND* to one packet.
- On each ACK for new data, increase *CWND* by one packet.
- When sending, send the minimum of the receiver's advertised window and *CWND*.

References

[1] G. R. Wright and W. R. Stevens, TCP/IP Illustrated, Volume I (The Protocols), *Addison Wesley, 1994.*

[2] K. Fall and S. Floyd., «Simulation-based comparison of Tahoe, Reno, and sack TCP», *in ACM computer communications review, July, 1996*.

[3] M. Zorzi and R.R. Rao., «Is tcp energy efficient?», *in Proceedings IEEE MoMuc, November 1999.*

[4] Ola Westin, «Performance issues in ad hoc networks», *29th November 2003.*

[5] J. Liu, S. Singh, «ATCP: TCP for Mobile Ad Hoc Networks», *IEEE journal on selected areas in communications, July 2001.*

[6] Kostas Pentikousis, «TCP in wired-cum-wireless environments», *4th quarter issue of IEEE Communications Surveys and Tutorials, 2000.*

[7] V. Jacobson., «Congestion avoidance and control», *SIGCOMM symposium on communications architectures and protocols, pages 314-329, 1988*. An updated version is available via ftp://ftp.ee.lbl.gov/papers/congavoid.ps.z.

[8] B. Wang, and S. Singh, «Computational Energy Cost of TCP», *ACM SIGMETRICS'03 Conference.*

[9] J. C. Hsiao-Keng, «Zero-Copy TCP in salaries», *in USENIX Annual Technical Conference, 1996, pp. 253-264.* [Online]. Available: citeseer.nj.nec.com/chu96zerocopy.html

[10] M. Allman, V. Paxon, W. Stevens, «RFC 2581: TCP Congestion Control», http://www.ietf.org/rfc/rfc2581.txt, *April 1999*.

[11] M. Zorzi and R. Rao., «Energy Efficiency of TCP in a local wireless environment», *Mobile Networks and Applications, Volume 6, Issue 3, July 2001.*

[12] V. Jacobson., «Modified TCP Congestion avoidance Algorithm», *Technical report, 30 Apr. 1990*. Email to the end2end-interest mailing list, URL ftp://ftp.ee.lbl.gov/email/vanj.90apr30.txt.

[13] J. Hoe., «Start-up Dynamics of TCP's Congestion Control and Avoidance Scheme », *June.1995. Master's thesis, MIT*.

[14] D.D. Clark and J. Hoe., «Start-up Dynamics of TCP's Congestion Control and Avoidance Scheme », *Technical report, June.1995. Presentation to the Internet end2end Research Group, cited for acknowledgement purposes only.*

[15] V. Tsaoussidis, A. Lahanas and C. Zhang, «The Wave and Probe Communication mechanisms», *Computer Science*.

[16] V. Ramarathinam, M. A. Labrador, «Performance Analysis of TCP over Static Ad Hoc Wireless Networks», *In Proceedings of the ISCA 15th International Conference on Parallel and Distributed Computing Systems (PDCS)", Pages 410-415, September 2002.*

[17] A. Chockalingam, M. Zorzi and R.R. Rao, «Performance of TCP on wireless fading links with memory», *in proc. IEEE globcom'93 (December 1993) PP. 542-549.*

[18] A. Chockalingam, M. Zorzi and R.R. Rao, «Performance analysis of TCP on channels with memory», *IEEE journal on Selected Areas in Communications (July 2000).*

[19] M. Zorzi and R.R. Rao, «Effect of correlated errors on TCP», *in proc. 1997 CISS (March 1997) PP. 666-671.*

[20] S. Agrawal and S. Singh, «An Experimental Study of TCP's Energy Consumption over a Wireless Link», *4th European Personal Mobile Communications Conference, Feb 20-22, 2001, Vienna, Austria.*

[21] H. Singh, S. Saxena, and S. Singh, «Energy Consumption of TCP in Ad Hoc Networks», *J. Wireless Networks, Vol. 10(5), Sep. 2004*.

[22] Michel M., Nelson L.S., and José F., «On the Performance of TCP Loss Recovery Mechanisms»,

[23] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, «TCP Westwood: Bandwidth Estimation for enhanced transport over wireless links», *proc. of the 7th annual international conference on mobile computing and networking, July 2001.*

[24] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J.–C. Chen, «A survey of energy efficient network Protocols for wireless Networks», *ACM/Baltzer Journal on Wireless Networks*, *vol. 7, No. 4, 2001, pp. 343-358.*

[25] K. Woo, C. Yu, D. Lee, H. Y. Youn, and Ben Lee, «Non-Blocking, Localized Routing Algorithm for Balanced Energy Consumption in Mobile Ad Hoc Networks», *MASCOTS'01*, Cincinnati, Ohio, Aug. 2001, pp. 117-124.

[26] M. Maleki, K. Dantu, and M. Pedram, «Power-aware Source Routing in mobile ad hoc networks», *Proceedings of ISLPED '02*, Monterey, CA, Aug. 2002, pp. 72-75.

[27] M. Maleki, K. Dantu, and M. Pedram, «Lifetime Prediction Routing in Mobile Ad Hoc Networks», *IEEE Wireless Communication and Networking Conf.*, Mars, 2003.

[28] S. Senouci, and G. Pujolle, «Energy efficient consumption in wireless ad hoc networks», *accepté à IEEE ICC'2004 (International Conference on Communications), Paris, Juin 2004*.

[29] IETF MANET WG (Mobile Ad hoc NETwork), www.ietf.ora/html.charters/manet-charter.html

[30] Elizabeth M. Royer, Chai-Keong Toh, «A Review of Current Routing Protocols for Ad Hoc Mobile Wireless Networks», IEEE Personal Communications, April 1999.

[31] C. E. Perkins and P. Bhagwat, «Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers», Comp. Comm. Rev., Oct. 1994, pp. 234-44.

[32] L. R. Ford Jr. and D. R. Fulkerson, «Flows in Networks», Princeton Univ. Press, 1962.

[33] C. E. Perkins and E. M. Royer, «Ad-hoc On-Demand Distance Vector Routing», Proc. 2nd IEEE Wksp. Mobile Comp. Sys. And Apps., Feb. 1999, pp. 90-100.

[34] D. B. Johnson and D. A. Maltz, «Dynamic Source Routing in Ad-Hoc Wireless Networks», Mobile Computing, T. Imielinski and H. Korth, Eds., Kluer, 1996, pp.153-81.

[35] J. Broch, D. B. Johnson, and D. A. Maltz, «The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks», IETF Internet draft, draft-ietf-manet-dsr-01.txt, Dec. 1998 (work in progress).

[36] A. Aaron and J. Weng, «Performance Comparison of Ad-hoc Routing Protocols for Networks with Node Energy Constraints», EE 360 Class Project, Spring 2000-2001.

[37] A. Huhtonen, «Comparing AODV and OLSR Routing Protocols», session on Internetworking, April 2004.

[38] Thomas Clausen, «Comparative Study of Routing Protocols for Mobile Ad-Hoc NETworks», INRIA, Mars 2004.

[39] Lawrence S. Brakmo, Sean W. O'Malley, and Larry L. Peterson «TCP Vegas: New Techniques for Congestion Detection and Avoidance», SIGCOMM, 1994.

[40] Network Simulator – NS-2. Available at www.isi.edu/nsnam/ns/

Additional Readings

[1] L. Feeny and M. Wrigh and M. Nilsson, «Investigating the energy consumption of a wireless network interface in an ad hoc networking environment», in Proceedings INFOCOM 2001, Anchorage, Alaska, 2001.

[2] M. Zorzi, M. Rossi, and G. Mazzini, «Throughput and energy performance of TCP on a wideband cdma air interface», in journal of wireless communications and mobile computing, Wiley 2002, 2002.

[3] S. Bansal et al., «Energy Efficiency and Throughput for TCP Traffic in Multi-Hop Wireless Networks», in Proceedings INFOCOM 2002, New York, NY, 2002.

[4] H. Singh and S. Singh, «Energy consumption of TCP Reno, New Reno, and SACK in multi-hop wireless networks», in ACM SIGMETRICS 2002, June 15-19 2002.

[5] V. Tsaoussidis et al., «Energy/Throughput Tradeoffs of TCP Error Control Strategies», proc. 5th IEEE Symp. Computers and Communications, France July 2000.

[6] M. Mathis, J. Mahdavi, S. Floyd, and A. Ramanow, «RFC 2018: TCP Selective Acknowledgement Options», http://www.ietf.org/rfc/rfc2018.txt, October 1996.

[7] J. Postel, «RFC 793: Transmission Control Protocol», http://www.ietf.org/rfc/rfc793.txt, September 1981.

[8] S. Floyd, T. Henderson, «RFC 2582: The New Reno Modification to TCP's Fast Recovery Algorithm», http://www.ietf.org/rfc/rfc2582.txt, April 1999.

[9] G. Holland and N. Vaidya, «Analysis of TCP Performance over Mobile Ad Hoc Networks», 5th annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'99), Seattle, Washington, August 1999.

[10] W. Stevens, «RFC 2001: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms», http://www.ietf.org/rfc/rfc2001.txt, January 1997.

[11] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, «RFC 2883: An Extension to the Selective Acknowledgement (SACK) Option for TCP», http://www.ietf.org/rfc/rfc2883.txt, July 2000.

[12] Robert R. Chodorek, and A. Glowacz, «Behaviour of TCP Westwood in Wireless Networks», Cracow, December 5-7, 2002, ATAMS 2002.

[13] C. Casetti, M. Gerla, Scott S. Lee, S. Mascolo, and M. Sanadidi, «TCP with Faster Recovery», Proceedings of the 7th annual international conference on Mobile Computing and Networking, Rome, Italy, p. 287-297, 2001.

[14] M. Gerla, Y. Sanadidi, R. Wang, C. Casetti, S. Mascolo, and A. Zanella, «TCP Westwood: Congestion Window Control Using Bandwidth Estimation», proc. Of IEEE Globecom 2001, S. Antonio, Texas, Dec. 2001.

[15] Joel Cannau, «Evaluation de performance d'une amélioration de TCP Westwood», en vue de l'obtention du Diplôme d'études approfondies en sciences appliquées, Année Académique 2002-2003.

[16] S. Senouci, «Application de techniques d'apprentissage dans les réseaux mobiles», Thèse de Doctorat de l'Université de Pierre et Marie Curie, PARIS, Octobre 2003.