TCP-WLAware: Wireless Loss-Aware TCP for IEEE 802.16 Networks

[1]K. SAKTHI MALA and [2]P .NAVANEETHAN

[1]Department of Computer Technology and Applications,
Coimbatore Institute of Technology
Coimbatore. 641 014.
INDIA.
[2]Professor & Head, Department of Electrical and Electronics Engineering,
PSG College of Technology,
Coimbatore 641 004.
INDIA.
E mail: [1]sakthimala@cit.edu.in, [2]pnn@eee.psgtech.ac.in

Abstract: - The performance degradation of TCP in wireless and wired-wireless hybrid networks is mostly due to lack of its ability to differentiate the packet losses caused by network congestions from the wireless transmission losses [17]. In IEEE 802.16 networks, wireless loss can occur due to wireless link errors and when the number of bandwidth requests exceeds the maximum. ARQ retransmits the blocks in these two cases repeatedly for a specified number of times till the expiry of block life time and drops the block if the transmission is not successful. Block life time and Request Retries for bandwidth requests are static parameters which are set by the service provider irrespective of the current Round Trip Time, wireless nature and network load. TCP has to retransmit the segment when the retransmission timer expires for such losses if the current window is small. This paper shows that TCP's throughput gets affected by the static parameters and proposes a cross layer feedback approach to enhance TCP-NewReno over IEEE 802.16 networks which can recover from such wireless packet losses and react without entering slow-start. The proposed scheme identifies the MAC block loss and informs the TCP sender about the loss which in turn resends the lost segment to which the block belongs before a time-out. This paper describes the design of the new proposed scheme, **TCP-WLAware** and presents results from experiments carried out using the NS-2 network simulator. The results from the simulations show that in a wireless congestion free environment, TCP-WLAware is able to apply TCP-NewReno's fast recovery at more number of instances than that of the TCP-NewReno without the enhancement and hence, recover more number of segments.

Key-Words: - Transmission Control protocol, TCP-NewReno, MAC, ARQ, Bandwidth requests, ARQ_BLOCK_LIFETIME, MAC layer feedback

1 Introduction

Wireless network technologies have evolved so as to provide flexible access to the Internet while moving. Wired networks provide the fixed point of network attachment, whereas wireless networks enable users to access the Internet from any place. There have been many attempts to replace the Internet access with WLAN and cellular networks. With increasing bandwidth, WLANs are successfully replacing wired networks in home and office environments. Due to the narrow coverage and lack of mobility support, however, they are not suitable for mobile users. On the other hand, cellular networks provide wider coverage and mobility support, and thus suitable for mobile users, but the communication cost and narrow bandwidth are the major obstacles to be widely deployed for the Internet access.

IEEE 802.16 WMAN technology [1] has been proposed to overcome the drawbacks of WLANs and cellular networks. IEEE standard 802.16 was designed to evolve as a set of air interfaces based on a common MAC protocol but with physical layer specifications dependent on the spectrum of use and associated regulations.

The operation of MAC services is connection-oriented. A connection is defined as a unidirectional mapping between BS and SS MAC peers for the purpose of transporting a service flow's traffic [1]. A service flow is a unidirectional flow of MAC

Protocol Data Units (PDUs) with predefined QoS parameters. The QoS parameters defined for the service flow are therefore implicitly provided by the connection's unique Connection Identifier (CID). In order to accommodate applications with different service requirements, the standard defines four types of MAC scheduling services namely, Unsolicited Grant Service (UGS), Real-time Polling Service (rtPS), Non-real-time Polling Service (nrtPS) and Best Effort (BE).

The UGS supports real-time uplink service flows that transport fixed-size data packets on a periodic basis, such as T1/E1 and Voice over IP without silence suppression. The rtPS is designed to support real-time UL service flows that transport variable-size data packets on a periodic basis, such as Moving Pictures Experts Group (MPEG) video. The nrtPS is designed to support delay-tolerant data streams consisting of variable-sized data packets for which a minimum data rate is required. The BE service is designed to support data streams for which no minimum service level is required and therefore may be handled on the basis of space availability.

Though, TCP based applications such as web browsing, email, FTP are classified as 'best effort', TCP's performance is sensitive to delay, jitter and packet loss. In a 802.16 network, where heterogeneous services and applications co-exist, resource allocation for best-effort applications may be limited [15] and hence TCP performance optimization is a research issue.

nrtPS and BE use bandwidth contention opportunities to send their bandwidth requests (BR). Due to link errors or instantaneous wireless network traffic, there are possibilities for these requests to be dropped after some retries. This leads to the drop of the data blocks for which the BR was raised. When the BR is successful, BS allocates Data Grant Burst Type IE, which is used to send the data block. This data block sent can even be dropped due to link errors. In both the cases, Automatic Repeat Request (ARQ) mechanism of 802.16 retransmits the dropped block few times until the expiry of ARQ_BLOCK_LIFETIME, and if the same condition persists, may give up the transmission.

The objective of this paper is to identify such drops and inform TCP sender to resend the block, which otherwise would have retransmitted the segment either after a time-out (TO) or triple duplicate acknowledgments (DUPACK). In this case, TCP would have considered TO or triple DUPACK as an indication of congestion and takes congestion recovery measures unnecessarily.

The rest of this paper is organized as follows. In Section II, we summarize IEEE 802.16 standard and present the motivation behind this work. Section III describes the ***TCP-WLAware*** mechanism. In Section IV, experimental results are presented and the efficiency of the scheme is compared with that of TCP-NewReno. We conclude the paper in Section V with a summary of the results and highlights of the future work.

2 Background and Motivation

2.1 TCP -NewReno

Reno TCP data sender retransmits a packet after a retransmit timeout has occurred, or after three duplicate acknowledgments have arrived triggering the Fast Retransmit algorithm [14]. A single retransmit time-out might result in the retransmission of several data packets, but each invocation of the Reno Fast Retransmit algorithm leads to the retransmission of only a single data packet.

In the case of multiple packets dropped from a single window of data, the first new information available to the sender comes when the sender receives an acknowledgment for the retransmitted packet. If there is a single packet drop and no reordering, then the acknowledgment for this packet will acknowledge all those packets transmitted before Fast Retransmit was entered.

However, if there are multiple packet drops, then the acknowledgment for the retransmitted packet will acknowledge some but not all the packets transmitted before the Fast Retransmit and this acknowledgment is termed as partial acknowledgment. TCP-NewReno provides an algorithm for responding to partial acknowledgments [13].

NewReno 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. In Reno, partial ACKs take TCP out of Fast Recovery by "deflating" the usable window back to the size of the congestion window (cwnd). In New-Reno, partial ACKs do not 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 the lost packets from that window have been retransmitted. NewReno remains in Fast Recovery until all of the data outstanding when Fast Recovery was initiated has been acknowledged.

Slow and steady variant of NewReno will reset the retransmission timer after each partial acknowledgment and retransmit the first segment. Hence, this variant will take N round trip times to recover N lost segments. Impatient variant of NewReno resets the retransmit timer only after the first partial ACK. In this case, if a large number of packets were dropped from a window of data, the TCP data sender's retransmit timer will ultimately expire, and the TCP data sender will invoke slow-start.

In IEEE 802.16 networks, when ARQ is enabled, a MAC SDU is logically partitioned into blocks. A MAC PDU may contain blocks that are transmitted for the first time as well as those being retransmitted. When a MAC PDU is dropped, the blocks present in the PDU, which may belong to different TCP segments, are dropped. Multiple packet drops due to errors are quite often possible in 802.16 networks. TCP-NewReno is an ideal option. But due to wireless losses, the size of the congestion window may not be enough to trigger fast recovery. Every time there is a TO, RTO gets doubled. Then to recover from occasional wired-loss, TCP sender needs to wait for a larger RTO. This may sometime leads to TCP disconnection.

2.2 Overview of IEEE 802.16

When a customer subscribes to the WiMAX service, has to provide the service provider the service flow information including the number of Uplink/Downlink (UL/DL) connections with the data rates and QoS parameters along with the kind of applications, he or she intends to run. The service provider will pre-provision the services by entering the service flow information into the Service Flow database [7].

When the Subscriber Station (SS) enters into the vicinity of the Base Station (BS) by completing the network entry and authentication procedure, the BS will download the service flow information from the Service Flow Database. When the SS registers with the BS by sending Registration Request (REG-REQ) message, the BS will be able to find the service flow information that has been pre-provisioned by using the MAC address of the SS. The BS will then use a Dynamic Service Addition (DSA) message to create service flows with the pre-provisioned service flow information. The service flows will then be available for the customer to send data traffic [7].

In IEEE 802.16 networks, a bandwidth request/grant mechanism is employed to acquire bandwidth for BE and nrtPS traffic. This mechanism is used for reducing data collision. When an SS has information to send and wants to enter the contention resolution process, it sets its internal backoff window equal to the request backoff start defined in the Uplink Channel Descriptor (UCD) message currently in effect.

The SS shall randomly select a number within its backoff window. This random value indicates the number of contention transmission opportunities that the SS shall defer before transmitting. These are defined by Request IEs in the UL-MAP messages.

After a contention transmission, the SS waits for a Data Grant Burst Type IE in a subsequent map. Once received, the contention resolution is complete.

The SS considers the contention transmission lost, if no data grant has been received in the number of subsequent Uplink map (UL-MAP) messages specified by the Contention-Based Reservation Timeout parameter. The SS now increases its backoff window by a factor of two, as long as it is less than the maximum backoff window. The SS randomly selects a number within its new backoff window and repeats the deferring process.

This retry process continues until the maximum number (request retries for Bandwidth Requests BR) of retries has been reached. At this time, the PDU shall be discarded [1]. If this request retries is very large, the connection queue may overflow.

802.16 provides error free communication by supporting ARQ. The ARQ is a control mechanism of data link layer where the receiver asks the transmitter to send again a block of data when errors are detected. The ARQ mechanism is based on acknowledgment (ACK) or non-acknowledgment (NACK) messages transmitted by the receiver to the transmitter to indicate a good (ACK) or a bad (NACK) reception of the previous frames.

ARQ maintains the following parameters [1]:
ARQ_RETRY_TIMEOUT is the minimum time interval a transmitter shall wait before retransmission of the unacknowledged block. The interval starts when the ARQ block was last transmitted.
ARQ_BLOCK_LIFETIME is the maximum time interval an ARQ block shall be managed by the ARQ transmitter state machine, once the initial transmission of the block has occurred. This is normally maintained as ARQ_RETRY_TIMEOUT * ARQ_RETRY_COUNT. If transmission of the

block is not acknowledged by the receiver before the time limit is reached, the block is discarded. Normally one timer is maintained which goes off every ARQ_RETRY _TIMEOUT. Blocks which are not acknowledged are resent. This will be repeated for a maximum retry count.
ARQ_RX_PURGE_TIMEOUT is the time interval the receiver shall wait after successful reception of a block that does not result in advancement of ARQ_RX_WINDOW_START, before advancing ARQ_RX_WINDOW_START.

ARQ transmission block state sequence [1] is shown in Fig .1.

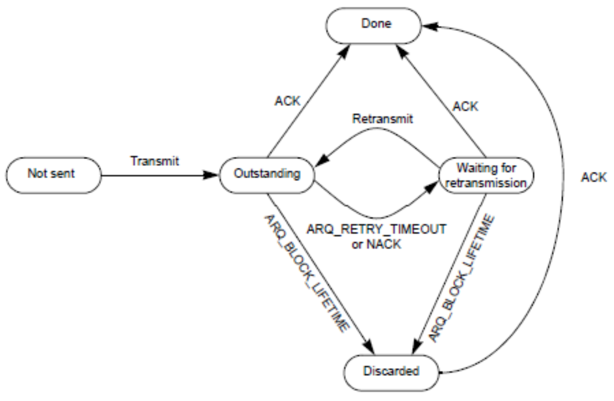

Fig.1 ARQ TX block state sequence

ARQ_BLOCK_LIFETIME is a crucial parameter which affects the upper layer's throughput. This parameter cannot be set to infinity as only finite delays and buffer sizes can be afforded in practice. TCP is designed to ensure reliable end-to-end transmission. TCP's performance is mainly determined by Round Trip Time (RTT) and loss rate. The RTT in the uplink direction is

$$RTT = (\max\left\{T^{i}_{notsent} \mid i \in B(j)\right\} + \max\left\{T^{i}_{arq} \mid i \in B(j)\right\} + T_{wire}) + (T_{wire} + T^{ack}_{notsent} + T^{ack}_{arq}) \quad \text{--(1)}$$

where,

$T^{i}_{notsent}$ is the duration for which the i^{th} block resides in notsent state.

$B(j)$ is the set of blocks that belongs to TCP segment, *j*.

T^{i}_{arq} is the duration for which i^{th} block is maintained by ARQ transmitter before successful transmission.

T_{wire} is the wired network delay.

$T^{ack}_{notsent}$ is the duration in which the ACK from receiver resides in the notsent state.

T^{ack}_{arq} is the duration for which the block carrying ACK block is maintained by ARQ transmitter before successful transmission.

When ARQ is enabled, a MAC SDU (an IP packet containing TCP segment) is sent as a set of ARQ blocks. The blocks may be carried in different MAC frames as per the bandwidth allotment. At the base station all the blocks are packed and then sent in the direction of the destination. Each block of the SDU will stay / maintained in notsent / by ARQ transmitter state chart for different periods of time. Hence, the maximum of them is considered in (1).

A MAC PDU drop results in the retransmission of a set of blocks. Any new block, *i* (as the result of a new TCP segment, j), will get a chance for transmission only after any such retransmissions. In this case there will be a raise in $T^{i}_{notsent}$. After sending once, if the block is dropped due to error or if there is a delay in bandwidth allocation T^{i}_{arq} will increase.

If all the blocks of the new TCP segment were transmitted within the ARQ_BLOCK_LIFETIME and if the TCP sender could receive an ACK for that segment before a time-out or 3 DUPACKs, the increased $T^{i}_{notsent}$ and/or T^{i}_{arq} will in turn slightly increase RTT. If all blocks were sent by the SS and the ACK is not received within RTO or 3 DUPACK, TCP will spuriously retransmit either after a time-out or 3 DUPACKs. In either case, cwnd will decrease, which reduces the TCP's goodput.

If any block is not sent within the lifetime, will cause the remaining blocks of the MAC SDU to be dropped which will cause TCP to retransmit the corresponding segment after a time-out.

Hence, a drop will affect the transmission of the subsequent blocks. Even though ARQ associated delays scales only sub-linearly, a sudden increase in $T^{i}_{notsent}$ and/or T^{i}_{arq} for any block will reduce the TCP's performance. [18] has presented measurement results from an early commercial deployment of a WiMAX-based broadband wireless access network. The authors have specified that many retransmissions occur in pairs and are two different segments, which are caused by real segment loss and not by delay variations due to ARQ. Actually the drop could be due to the delay caused by ARQ in transmitting the previous blocks.

This paper has studied the performance of one FTP connection over 802.16 by varying the network load. We have made use of Qualnet 4.5 evaluation copy to construct 802.16 network with one cell and modeled the wireless channel with Rayleigh fading. The frame structure has 5ms frame length, 10MHz bandwidth with 1024 FFT. The transport protocol chosen is TCP-NewReno and the maximum segment size chosen is 512 bytes. In this work, we present results based on a terminating simulation with a fixed simulation time.

Setting larger ARQ_BLOCK_LIFETIME doesn't improve the FTP throughput. The simulation also shows that FTP throughput gets maximized at different ARQ_BLOCK_LIFETIME value for different loads as shown in Fig.2 and Fig.3

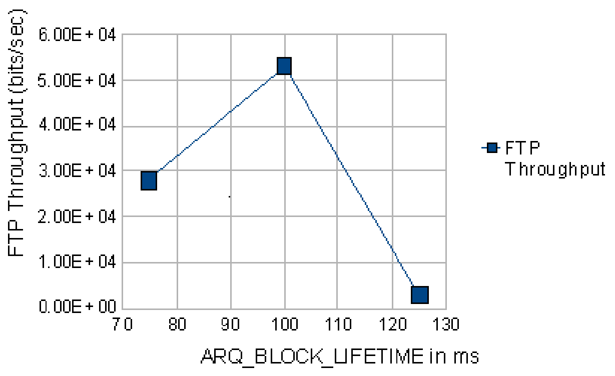

Fig.2 FTP throughput Vs ARQ_BLOCK_LIFETIME when no other load is present

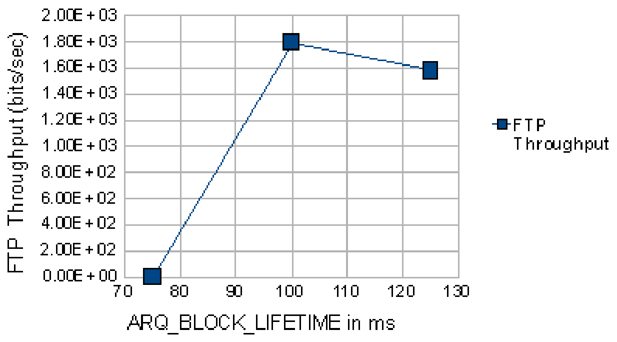

Fig.3 FTP throughput Vs ARQ_BLOCK_LIFETIME when 10 CBR traffic connections are active

Hence, ARQ parameters need to be adjusted dynamically based on the current network condition, load, and RTT to improve the performance of TCP or the loss due to wireless error need to be recovered before TCP recovers the lost segment by applying congestion recovery procedure.

2.3 Related work

Several schemes ([11], [22], [8]) have been proposed to alleviate the effects of non-congestion related losses on TCP performance in error prone wireless networks.

The base for all such works is Explicit Loss Notification proposed in [22]. This scheme provides a way by which senders can be informed that a loss happened because of reasons unrelated to network congestion so that sender retransmissions can be decoupled from congestion control. This information is indicated by the BS in ACK segments arriving from the receiver. Even though the loss has happened in the wireless hop, this loss information has to wait till there is an ACK from the receiver specifying that this missing segment is expected. The chance for entering slow start remains the same as that of the TCP without ELN.

In Link-Layer-originated Explicit Link Status Notification (LL-ELSN) scheme [8], BS sends Explicit Retransmission Start Notification (ERSN) to the TCP sender when the first transmission attempt fails for a packet. Upon reception of ERSN, the TCP sender neither invokes congestion control due to the packet nor retransmits it. When the packet is discarded by the station, Explicit Loss Notification (ELN) message is sent. On receiving ELN, the sender retransmits the missing segment. LL-ELSN uses the SACK information to recover multiple losses. To investigate the performance of the scheme, 802.11b is used as the link layer. But in [4] authors have shown using simulations that SACK consumes more computational energy in all packet loss situations than New-Reno in MANETs. Processing the feedback from MAC will worsen the computational energy as TCP has to maintain and process the states of the various segments. Moreover, the author has not mentioned about the handling of RTT measurement, when active for the lost packet.

Many researchers have presented various approaches to improve the performance of TCP over 802.16 networks.

In [5], the authors propose an ACK Unifier and an extractor, to reduce the drop rate or delay of the ACK packet between MS and SS. When a TCP ACK reaches MAC, the unifier searches for the TCP packet with payload and copies the ACK information to the selected packet and discards the ACK packet. Extractor at the other end reconstructs the ACK packet and sends it. Both the ends need to calculate the checksum. However, when duplicate ACK are unified, it is not possible to extract the actual number of duplicates. Then invoking fast recovery for the lost segment will not be possible.

A new transmission scheme, where the ACK packets are combined with Bandwidth Request (BR) header is proposed by the authors in [6]. This scheme cannot be used during piggybacked ACK on data segment.

In another related work [9], authors proposed a new scheme where, feedback about the channel state is sent to the TCP, which then adaptively control packet size, packet amount and retransmission decision (Jin 2008). The channel state is measured in terms bit error rate.

Oleg G Ivanov et al. has suggested that an adaptive ARQ is required for WiMAX networks in [12]. Similarly, in [3] authors have shown that frame duration, direction of flow, DL:UL ratio, MCS and offered load affect the performance of TCP. Through simulations [21] has revealed that ARQ and its configuration play an important role in data transmission. They have not specified about the effect of the static parameters.

The proposed scheme modifies TCP-NewReno to recover multiple losses. TCP-NewReno can be applied to connections that are unable to use the TCP -SACK option [13]. Our work is motivated by previous studies that indicate TCP-NewReno is the widely deployed non-SACK loss recovery strategy in Internet [10].

3 TCP-WLAware

It is clear that an efficient mechanism is required to improve the performance of TCP irrespective of the chosen MAC layer parameters and RTT. TCP-WLAware is one such novel scheme which enhances TCP-NewReno by providing a feedback about the loss by exploiting cross layer design. This design violates the layered architecture by creating new upward interface [16] to send the sequence number of the lost segment to TCP at runtime.

An ARQ block may be dropped due to either repeated link errors and/or expiry of the request retries for Bandwidth Requests.

IEEE 802.16 document specifies that a Discard message shall be sent following violation of *ARQ_BLOCK_LIFETIME*. The message may be sent immediately or may be delayed up to *ARQ_RX_PURGE_TIMEOUT* + A*RQ_RETRY_TIMEOUT*. When a discard message is received from the transmitter, the receiver shall discard the specified blocks, advance A*RQ_RX_WINDOW_START* to the Block Sequence Number (BSN) of the first block not yet received after the BSN provided in the Discard message, and mark all not received blocks in the interval from the previous to the new *ARQ_RX_WINDOW_START* values as received for ARQ Feedback IE reporting [1]. On receiving ARQ Feedback IE, transmitter sets *ARQ_TX_NEXT_BSN* to the new block number to be sent.

Delaying the Discard message by *ARQ_RX_PURGE_TIMEOUT+ARQ_RETRY_TIMEOUT* will cause unnecessary delay and requires a timer to run for each block, which is costly.

TCP-WLAware sends the Discard message as soon as the *ARQ_BLOCK_LIFETIME* expires. The *ARQ_BLOCK_LIFETIME* may expire between the arrival of block at the receiver and its ARQ feedback at the transmitter. When the discard message is sent immediately, the feedback from the receiver will not convey as to whether the block is really discarded or not As TCP-WLAware need to send a feedback to TCP sender about the loss, first it has to confirm the loss. This paper defines a new ACK type, referred to as ***Selective NACK*** which can be sent after receiving a discard message when the block is really discarded. ARQ feedback IE with this new type can be sent.

When a discard message is received, the receiver checks if the block is already received. If so, the receiver sends *ARQ_FEEDBACK_IE* with ***selective ACK***. Otherwise, ARQ receiver removes the received other fragments of the SDU from the queue and then sends a ***NACK of type 5*** to confirm the discarding which is shown in Table 1. The modified ARQ TX block state sequence is shown in Fig 4.

Table 1 ARQ feedback IE format with the added new type

Syntax	Size (bits)	Notes
ARQ_Feedback _IE(LAST) {	variab le	
CID	16	The ID of the connection being referenced
LAST	1	0 – More ARQ Feedback IE in the list 1 – Last ARQ Feedback IE in the list
ACK Type	3	0x0 – Selective ACK entry 0x1 – Cumulative ACK entry 0x2 – Cumulative with Selective ACK entry 0x3 – Cumulative ACK with Block Sequence ACK entry ***0x5 – Selective NACK entry***

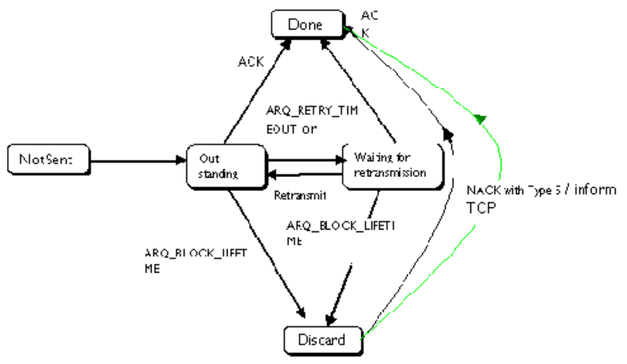

Fig.4 Modified ARQ TX block state sequence

On receiving ***NACK of type 5***, the MAC sender sends a feedback to the TCP sender. For uplink traffic, the TCP sender is in the local host and for downlink traffic, TCP sender in located in the actual source host. The feedback contains two fields, arq_retransmit, and arq_seq_. arq_retransmit is a flag that indicates the segment loss and arq_seq_ holds the sequence number of the lost segment. TCP-WLAware uses the reserved bits and sequence number fields present in the TCP header to convey the feedback.

TCP sender then enters MAC drop recovery procedure. The modifications to the TCP-NewReno algorithm at the various steps are:

1. If the received packet contains an indication that a packet is dropped by the MAC layer and if the sequence number is greater than last received acknowledgment, store the sequence number of the dropped packet in ***"arq_recover_list_"*** and the current instance of time in ***"arq_recover_time_list"*** and resend the segment lost. If RTT measurement is active for this segment, disable it. Reset the retransmission timer.
2. When invoking fast retransmit for any of the packets in the ***arq_recover_list_***, all the duplicate acknowledgments must have arrived after the respective ***arq_recover_time_list*** + ***rtt***. When all such duplicates have been received, remove the segment from the ***arq_recover_list_*** and then invoke fast retransmit.
3. When a full ACK arrives, that acknowledges new data, clear the entries in ***arq_recover_time_list and arq_recover_list_*** that are covered by the new acknowledgment and exit the MAC drop recovery procedure, if the list is empty.
4. When a partial acknowledgment is received, remove the entries in the ***arq_recover_time_list and arq_recover_list_*** that are covered by this acknowledgment and invoke the partial acknowledgment procedure, only if the first unacknowledged segment is not in the ***arq_recover_list_***.
5. After a retransmit timeout, if the sender is in the MAC drop recovery procedure, exit clearing ***arq_recover_list_& arq_recover_time_list.***

4 Simulation Results

We used NS-2 WiMAX Simulator Release 2.6 provided by the WiMAX forum [20] to construct a WiMAX network with one cell. This version tries to send a block, infinite number of times and hence no discards. To minimize delay and buffer sizes, truncated ARQ has been adapted in practice. Hence we extended the simulator by making the MAC layer to drop the block after the violation of ARQ_BLOCK_LIFETIME. Instead of modelling WiMAX's timer based ARQ retransmission management, we considered a maximal retransmission count, *arq_max_retries*. This paper studies the change in the congestion window (cwnd) size, as cwnd indicates if fast recovery is applied. The parameters used in the simulation are reported in Table 2 and the simulation topology is shown in Fig.5.

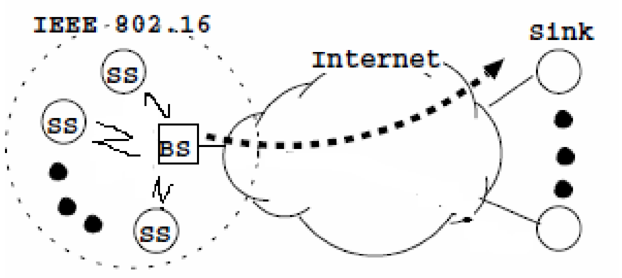

Fig. 5 Simulation topology

CASE 1: TCP -NewReno / TCP-WLAware without wired errors

First we observed the change in cwnd with time with TCP-NewReno as the sender agent. The variation of cwnd is shown in Fig.6. There are no drops in the wired network and wireless loss model uses a uniform distribution with an error rate of 0.15 to randomly drop packets. There had been only one instance of evoking NewReno's fast recovery in the total duration of 170 sec. Initially, when the cwnd is small, the loss of segments 0, 2, and 3 causes time-out and TCP enters slow-start.

Table 2 Simulation parameters

Parameter	Value
Frame duration	5ms
packet_size	128 bytes
arq_block_size_	32 bytes
arq_max_retries	7
arq_retrans_time	80 ms
PHY	OFDMA
Bandwidth	10MHz
FFT	1024
Cyclic Prefix length	1/4
Modulation and coding	QPSK ¾
bw_req_contention_size_	5
request_retry_	2
t16_timeout_	100ms
TCP Version	NewReno
Delayed ACK factor	1
NewReno variant	slow-but-steady

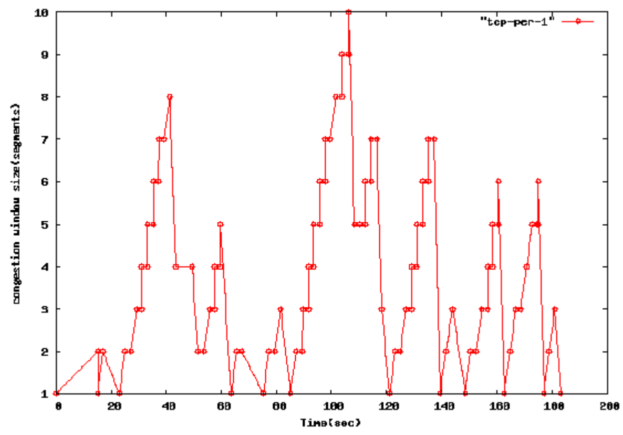

Fig. 6 The variation of cwnd over time

Fig. 7 shows the segments sent and lost during the simulation period. At 39.75 sec, segments 36 through 38 were dropped. These segments were all sent in one physical frame and was dropped once due to error. After the ARQ retransmission time expires, MAC puts them in the retransmission queue. Before getting a chance for transmission, these blocks were dropped due to the expiry of retransmission count. At 43.24 sec, the loss of segment 36 causes TCP to enter fast recovery as the window at that point of time is positioned at 8. As the segments 37 and 38 were also dropped at the same time, TCP recovers the lost segments one per *rtt* applying NewReno. At 45.75 sec, segment 47 was dropped. As the cwnd was 4, there were enough duplicate ACKs, fast recovery was evoked. At 63.48 sec, TCP enters slow-start as the segments 62-66 were dropped. As large number of segments in a row was dropped, enough duplicates were not there to trigger fast recovery.

This can frequently happen in 802.16 networks as blocks from different segments may be packed in one MAC PDU. The same scenario has happened at 139.40 sec and 162.68 sec.

Secondly, we observed that there was a change in cwnd, when TCP-NewReno with WLAware is chosen as the agent. In this case , whenever there is a block drop, ARQ identifies and informs the loss to TCP. So the congestion window grows which maximizes the throughput. The variation of cwnd in this case is shown in Fig. 8.

CASE 2: TCP -NewReno /TCP-WLAware with wired errors

Then we introduced packet loss in the wired network and studied the variation in congestion window with TCP-NewReno as the agent. The wired loss model uses uniform distribution with packet loss rate of 0.1. Fig.9 shows the variation

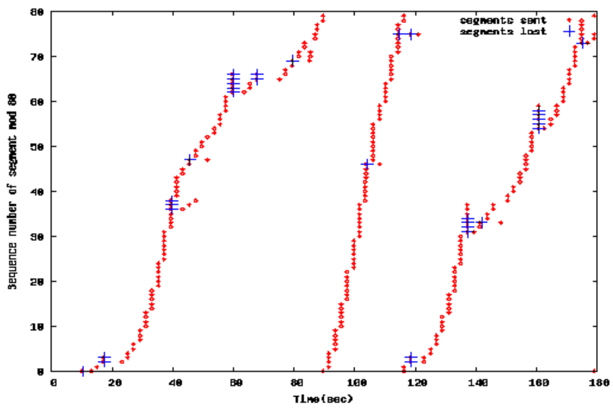

Fig. 7 Sequence number of the segments sent with the lost segments

of cwnd when WLAware is not enabled and with wired errors. Due to wireless losses, sender often enters slow-start. Under these conditions, when a segment is lost due to congestion or error in the wired infrastructure, there won't be enough duplicates to trigger fast retransmission. So time-out is the option for retransmission in most situations.

Segments 8, 9 and 10 were sent in one MAC frame and dropped due to error at 68.07 sec. Before they get a chance for retransmission, they were dropped due to the expiry of retry count. The congestion window at that time was 3 and as all the three segments were dropped, there were no duplicates to trigger NewReno. Hence, the cwnd drops down to 1. At 138.44 sec, segment 59 was dropped in the wired network. At 139.75 sec, segments 60, 61 were dropped in the wireless hop. As the cwnd size at that point of time was 3, there were no outstanding segments to trigger fast recovery.

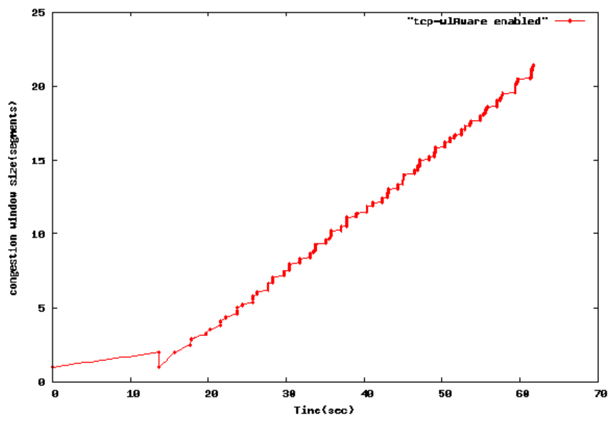

Fig. 8 Variation in cwnd when TCP-WLAware is enabled.

At 157.75 sec, segment 88 was dropped in the wireless hop. At 159.20 sec, wired network dropped segment 91. As the cwnd size was 8, TCP could recover segments 88 and 99 by applying NewReno's fast recovery. This is the only instance when NewReno could be applied during the simulation period.

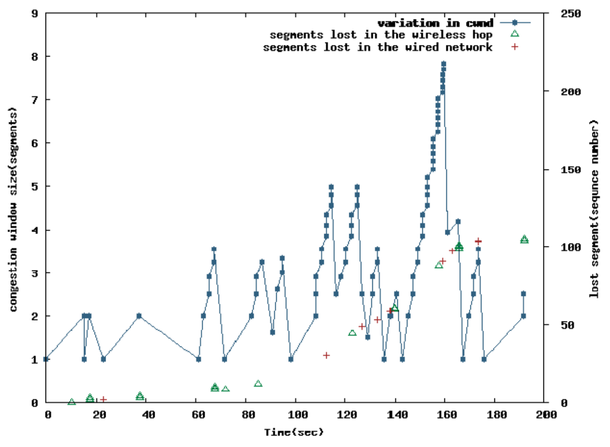

Fig.9 Change in congestion window with wired loss when WLAware is not enabled

Then, we studied the performance of TCP-WLAware in the presence of wired errors. TCP recovers from all wireless losses and allows the congestion window to grow. This growth allows NewReno to be applied for wired losses. The variation in congestion window is shown in Fig.10. Wireless drops do not make any alteration to the cwnd.

TCP-WLAware facilitates NewReno to be applied 4 times. At 133.261 sec, 3 DUPACKs were received for 123 and TCP applies fast recovery and recovers 123. TCP remains in fast recovery and receives partial ACK for 127. This triggers NewReno's fast recovery so as to recover 127. NewReno's fast recovery is also applied to recover segments 138, 162, and, 204 in a similar fashion.

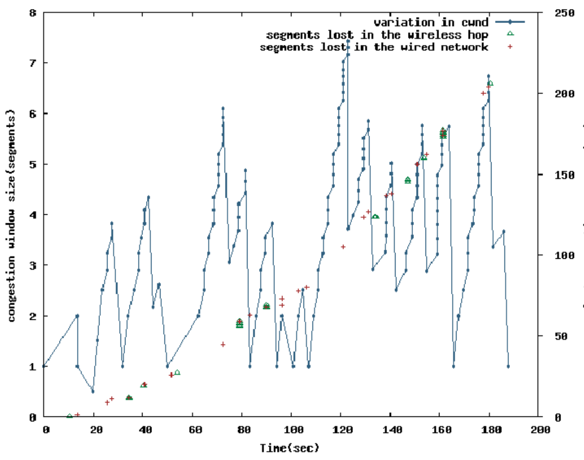

Fig.10 Variation in the cwnd with wired and wireless errors when WLAware is enabled

All the other dips in Fig.10 depict the normal behavior of NewReno. At 25.67 sec, segment 9 dropped in the wired hop and hence the cwnd is set to 1. Similarly, loss of segments 73 and 78 causes slow-start. In these cases, the congestion window size was less than 2.

At 161.69 sec, segments 173-177 were dropped in the wireless hop and were transmitted by resetting the retransmission timer. Segment 174 was dropped again in the wired hop. TCP-WLAware considers the duplicates arriving after the transmission of the lost segment. Even though the cwnd is greater than 4, as there are no enough duplicates to trigger NewReno, slow-start begins. The same scenario has got repeated when segments 26, 59, and 68 were dropped.

The losses in the wireless hop were recovered immediately without delay and the efficiency of NewReno is only dependent on the wired errors.

TCP -WLAware allows NewReno to be applied in all possible wired loss scenarios by hiding the wireless losses. Hence, the average number of packets sent by TCP-WLAware will be more than that of the TCP-NewReno.

5 Conclusion

In this paper, we have measured the throughput of one FTP connection at various network loads and found that the maximum throughput is achieved at different ARQ_BLOCK_LIFETIME for each network load. Making this parameter static may cause TCP to spuriously retransmit or time-out often. This paper has proposed a new cross layer feedback mechanism, which identifies wireless losses and inform TCP. TCP recovers such losses immediately, which normally would lead to slow-start. This work has presumed that there is no congestion within the wireless network, and hence the drops are only due to errors. This work can be extended by considering the wireless drops due to congestion. Our future research is to improve TCP-WLAware to distinguish wireless congestion losses from losses caused by errors. If the loss is due to congestion, then congestion control measures need to taken.

References:

[1] IEEE 802.16-2005. IEEE Standard for Local and Metropolitan Area Networks-Part 16: Air Interface for Fixed Broadband Wireless Access Systems, 2006.

[2] Richard Stevens. W. TCP/IP Illustrated, Volume 1- The Protocols . Pearson Education, 2005.

[3] Addisu Eshete, Andrés Arcia , David Ros and Yuming Jiang. Impact of WiMAX Network Asymmetry on TCP , *Proceedings of the IEEE Wireless communications & Networking conference*, 2009, 1706 – 1711.

[4] Alaa Ghaleb-Seddik , Yacine Ghamri-Doudane , Sidi Mohammed Senouci , Nazim Agoulmine. Measurement of TCP computational and communication energy cost in MANETs. *Pervasive and Mobile computing*, 2011,7(1): 60-77.

[5] Bong-Ho Kim, Jungnam Yun, Yerang Hur, Chakchai So-In, Raj Jain & Abdel -Karim Al Tamimi. Capacity Estimation and TCP Performance Enhancement over Mobile WiMAX Networks. *IEEE communications Magazine*, 2009, 47(6):132-141.

[6] Byongkwon Moon & Jeonghoon Mo. Optimizing uplink TCP-ACK transmission in WiMAX OFDMA systems. *IEEE Communication Letters*, 2008, 12(4): 256- 259.

[7] Govindan Nair et al. IEEE 802.16 Medium Access Control and Service Provisioning. *Intel Technology Journal*, 2004, 8(3), 213-228.

[8] Ji-Hoon Yun. Cross-Layer Explicit link Status Notification to improve TCP Performance in Wireless Networks. *EURASHIP Journal on Wireless Communication and Networking*, 2009, 1-15.

[9] Jin Hwang, Sang Woo Son, & Byung Ho Rhee. Improving TCP Performance over WiMAX Networks Using Cross-Layer Design. *Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology*, 2008(2) : 83-88.

[10] Medina, A. , Allman, M. , & Floyd, S. Measuring the evolution of transport protocols in the Internet. *ACM SIGCOMM Computer Communication. Rev.*, 2005, 35(2), 37-52.

[11] Xu K., Tian Y., & Ansari N. TCP-Jersey for Wireless IP Communications. *IEEE Journal on Selected Areas in Communications*, 2004, 22(4), 747-756.

[12] Oleg G Ivanov, Andrey S Bazhenov. Adaptive ARQ for WiMAX networks. *Proceedings of the IEEE International Conference*, 2009, Eurocon, 1284-1287.

[13] Floyd S., Henderson T., et al. (2004). RFC 3782: The NewReno Modification to TCP's Fast Recovery Algorithm. IETF, http://www.ietf.org/rfc/rfc3782.txt

[14] Allman, M.,V. Paxson & Stevens, W. RFC 2581 TCP Congestion Control.1999, IETF, http://www.ietf.org/rfc/rfc25581.txt

[15] Xiang ying Yang, Muthaiah Venkatachalam, & Shantidev Mahanty. Exploiting the MAC layer flexibility of WiMAX to systematically enhance TCP performance. *Proceedings of the Mobile WiMAX symposium*,2007, 60-68.

[16] Vineet Srivatsava, & Mehul Motani. (2005). Cross-Layer Design: A survey and the road ahead", *IEEE Communication Magazine*, 112-119.

[17] Ye Tian, Kai Xu, & Nirwan Ansari. TCP in wireless Environments: Problems and solutions. *IEEE Radio Communications*,2005, 527-532.

[18] Emir Halepović, Qian Wu, Carey Williamson, Majid Ghaderi. TCP over WiMAX: A Measurement Study. *IEEE International symposium on Modeling, Analysis and simulation of computers and Telecommunication systems*, MASCOTS 2008, 1-10.

[19] NS. The Network Simulator-ns-2, http://www.isi.edu/nsnam/ns.

[20] WiMAX patch. The WiMAX Forum System Level Simulator NS-2 MAC+PHY Add-On for WiMAX (IEE802.16), http://code.google.com/p/ns2-wimax-awg/downloads/list

[21] Sayenko .A, Tykhomyrov.V, Martikainen. H, & Alanen.O. Performance analysis of the IEEE 802.16 ARQ mechanism. Proceedings of the 10th ACM Symposium on Modeling, analysis, and simulation of

wireless and mobile systems,2007, 314-322.
[22] Hari Balakrishnan and Randy H. Katz. Explicit Loss Notification and Wireless Web Performance,1998, Proc. IEEE Globecom Internet Mini-Conference, Sydney, Australia.

K.Sakthi mala received the B.Sc degree in Physics from Bharathiar University, India in 1986 and M.Sc Applied Physics and Computer Electronics from Bharathidasan University, and M.Phil in Computer science from Bharathiar University, India. She is working towards the Ph.D degree in Science and Humanities at Anna University Chennai, India. Her current research interests include wireless TCP and Wireless MAN.

Dr. P. Navaneethan received the BE in Electrical and Electronics degree from University of Madras in 1981and ME degree in Applied Electronics from Anna University in 1983 and PhD in Computer science from Indian Institute of Science, Bangalore in 1991. He is currently working as Professor & Head of Electrical And Electronics Engineering Department, PSG College of Technology, India. He had presented about 15 International and National conference papers. His current research interests include the various aspects of computer networks, computer architecture and multilingual computing.