Poster Abstract: INET framework extensions for TCP Vegas and TCP Westwood

María Fernández, Carlos T. Calafate, Juan-Carlos Cano, Pietro Manzoni
Department of Computer Engineering
Universitat Politècnica de València (UPV)
Camino de Vera, S/N - 46022 Valencia, Spain
maferhe2@alumni.upv.es, {calafate, jucano, pmanzoni}@disca.upv.es

ABSTRACT

Currently, the OMeT++ simulation platform lacks novel enhancements to the TCP protocol targeting wireless environments, offering just the standard TCP implementation. This is a significant drawback when compared to other widely adopted simulation tools - e.g. ns-2, ns-3, Opnet -, especially when taking into consideration the relevance of wireless technologies in current and future research efforts in the networking area. In an attempt to mitigate this gap, we have implemented two well-known TCP variants for the OMNeT++ platform, which are described, validated and evaluated in this paper. Our main goal is to provide a basic framework for the development of new TCP implementations, mostly oriented to wireless channels, by offering a set of well-established reference protocols (Reno, Vegas, and Westwood) according to current literature. Simulation experiments under different channel delay and loss conditions are used to validate the implemented protocols, as well as to provide reference performance levels for other related proposals to be compared against.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Modeling techniques; I.6 [**Simulation and modeling**]: Model Development—*Modeling methodologies*; B.4 [**Input/output and data communications**]: Performance Analysis and Design Aids—*Simulation*

General Terms

Performance, Design

1. INTRODUCTION

TCP is one of the keystones in the Internet architecture. Since its creation in the early 70's, it has become, together with the IP protocol (TCP/IP), the global communications standard for computer networks [7]. Over the years, much research has been done, and many improvements have been proposed in the scope of traditional wired networks [8, 9, 4]. However, as part of the network development progress, different kinds of high-speed wireless networks have also emerged [11]. This meant that the TCP protocol, despite being originally designed for wired environments alone, was also adopted for wireless communications over the past decade, including both heterogeneous networks (with wired and wireless sides), and purely wireless networks.

TCP Reno, derived from TCP Tahoe, is considered the main reference implementation, being the basis of many of the current systems [12]. However, since the design of TCP's congestion control mechanism is optimized for wired networks, it does not provide the same performance in wireless environments since the characteristics of the latter differ significantly from the former. Basically, packet losses are associated with congestion in wired networks, causing the amount of packets injected into the network to be reduced; in wireless networks, losses are usually related to problems in the wireless channel itself, and reducing the packet transmission rate is not the adequate action to take.

To address the limitations of standard TCP in wireless channels, different solutions were proposed over the last years [1, 3, 2, 6]. Nevertheless, no solution has yet become a *de facto* replacement for TCP in current devices, and this issue remains an open topic, especially in novel wireless research areas [11] such as 4G/5G networks or vehicular networks.

Currently, the OMNeT++ simulation platform includes a standard TCP implementation. However, when addressing the challenging research issues of upcoming wireless technologies, the availability of other TCP variants becomes quite relevant to the research community in order to provide some reference performance levels to compare against. To this end, we have implemented two different alternatives: TCP Vegas [4], which is a widely adopted variant of TCP used as reference in many research papers, and TCP Westwood [5], a proposal that specifically addresses the challenges of wireless environments. Simulation experiments show that the results obtained agree with other published results.

The paper is organized as follows: in section 2 we provide an overview of TCP Reno, along with the two TCP variants implemented: Vegas and Westwood. OMNeT++ implementation details are provided in section 3. Experimental results are then presented in section 4 to validate the implementations, providing some reference performance values. Finally, section 5 concludes the paper.

OMNeT Workshop 2013, March 05-07

DOI 10.4108/icst.simutools.2013.251587

2. OVERVIEW OF TCP RENO, VEGAS AND WESTWOOD

The Transmission Control Protocol (TCP), initially defined in RFC 793 [10], is a standard communications protocol from the transport layer. In practice, it is included in every TCP/IP implementation that is not used exclusively for routing. TCP provides reliable and ordered data transfers due to the use of acknowledgments (ACKs), sequence numbers and error detection; additionally, it also offers a connection-oriented communication, flow control, and multiplexing by introducing the concept of *port*. TCP Reno is considered the standard TCP implementation, and it relies on four basic algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery.

2.1 TCP Vegas

TCP Vegas [4] is an implementation based on modifications to the Reno implementation. It does not involve any changes to the TCP specification; it is merely an alternative implementation that interoperates with any other valid implementation of TCP. In fact, all the changes are confined to the sending side.

TCP Vegas employs three proactive techniques to increase throughput and decrease losses. The first technique, a new retransmission mechanism, results in a more timely decision to retransmit a dropped segment; in particular, it uses a more accurate RTT estimation to decide whether to retransmit a segment whenever an ACK is received. The second technique gives TCP the ability to anticipate congestion, and adjust its transmission rate accordingly. The algorithm tries to maintain the "right" amount of extra data in the network, where by extra data we mean data that would not have been sent if the bandwidth used by the connection exactly matched the available bandwidth in the network. The final technique modifies TCP's slow start mechanism so as to avoid packet losses, while trying to find the available bandwidth during the initial use of slow start.

2.2 TCP Westwood

TCP Westwood [5] attempts to enhance the performance of TCP's congestion control window based on the feedback provided by end-to-end bandwidth measurements. The available bandwidth is estimated at the TCP source by measuring and low-pass filtering the returning rate of acknowledgments. The estimated bandwidth is then used to properly set both the congestion window and the slow start threshold after a congestion episode, that is, after a timeout or 3 duplicate acknowledgments. The rationale of this strategy is simple: TCP Westwood sets a slow start threshold and a congestion window which are consistent with the network capacity measured at the time congestion is experienced. In particular, TCP Westwood introduces a new mechanism of faster recovery to avoid an overly conservative reduction of the congestion window after a congestion episode by taking into account the end-to-end estimation of the available bandwidth. The advantage of the proposed mechanism is that the TCP sender recovers faster after losses, especially over connections with large round trip times, or when running over wireless links, where sporadic losses are due to unreliable links rather than congestion.

Algorithm 1 TCP Vegas: RTT and Timeout calculation.

```
void TCPVegas::receivedDataAck(uint32 firstSeqAcked) {

      TCPBaseAlg::receivedDataAck(firstSeqAcked);

      simtime_t tSent = state->v_sendtime[(firstSeqAcked - (state->iss+1))
% state->v_maxwnd];
      simtime_t currentTime = simTime();

if (tSent != 0 && num_transmits == 1) {
      simtime_t newRTT = currentTime - tSent;
      state->v_sumRTT += newRTT;
      ++state->v_cntRTT;

      if (newRTT > 0) {
            if(newRTT < state->v_baseRTT)
                  state->v_baseRTT = newRTT;

            simtime_t n = newRTT - state->v_sa/8;
            state->v_sa += n;
            n = n < 0 ? -n : n;
            n -= state->v_sd / 4;
            state->v_sd += n;
            state->v_rtt_timeout = ((state->v_sa / 4) + state->v_sd) / 2;
            state->v_rtt_timeout += (state->v_rtt_timeout / 16);
      }
}
```

3. IMPLEMENTATION OF TCP VEGAS / WESTWOOD IN OMNET++

TCP Vegas and TCP Westwood have been implemented[1] based on the existing TCP Reno code available in the INET 2.0 Framework. Since these versions only involve modifications at the sender side (in the control congestion algorithm) with respect to Reno, it has been determined that both *TCPVegas* and *TCPWestwood* classes should extend the *TCPBaseAlg* class, similarly to the *TCPTahoeRenoFamily* class.

3.1 TCP Vegas

In the *TCPVegas* class, a new method (*checkRTTTimer*), had to be defined. Additionally, methods *recalculateSlowStartThreshold, processRexmitTimer*, *receivedDataAck*, *receivedDuplicateAck,* and *dataSent* had to be redefined without losing the basic functionality that the *TCPBaseAlg* class methods offer.

One of the most important features of Vegas is the more accurate RTT measurement mechanism. In this implementation, whenever a packet is sent, its sending time is registered by the *dataSent* method using vector *v_sendtime* along with the *rcv_wnd* size (max. capacity of the receiver buffer). With these timestamps and the current time, the RTT of a segment and the timeout that Vegas proposes (*v_rtt_timeout)* can be easily calculated when an ACK is received (see Alg. 1).

Based on the calculated RTT value, the new retransmission mechanism is carried out by the *receivedDataAck* and *receivedDuplicateAck* methods. These methods rely on the *checkRTTTimer* method to check whether the time since the oldest unacknowledged segment (*snd_una*) exceeds the Vegas timeout value.

It is also important to highlight that the congestion window and the slow start threshold are adjusted by the *receivedDataAck* method, using the difference between the expected and the actual throughput. To this purpose it relies on the three thresholds defined by Vegas, *v_alpha*, *v_beta* and *v_gamma*, to maintain the "right" amount of extra data in the network [4].

[1]Available at: https://github.com/maferhe2/TCP-Vegas-Westwood

Algorithm 2 The *recalculateBWE* method in TCPWestwood.

```
void TCPWestwood::recalculateBWE(uint32 cumul_ack) {
    simtime_t currentTime = simTime();
    simtime_t timeAck = currentTime - state->w_lastAckTime;

    // Update BWE
    if(timeAck > 0) {
        double old_sample_bwe = state->w_sample_bwe;
        double old_bwe = state->w_bwe;
        state->w_sample_bwe = (cumul_ack) / timeAck;
        state->w_bwe = 0.9047*old_bwe +
                       0.0476*(state->w_sample_bwe + old_sample_bwe);
    }
    state->w_lastAckTime = currentTime;
}
```

3.2 TCP Westwood

In *TCPWestwood*, methods *dataSent*, *recalculateSlowStartThreshold*, *processRexmitTimer*, *receivedDataAck* and *receivedDuplicateAck* had to be redefined, without losing the functionality of the *TCPBaseAlg* class methods. Additionally, method *recalculateBWE* was introduced, being the core of the TCP Westwood implementation. This method is in charge of doing bandwidth estimations based on the received ACKs ratio (*cumul_ack*) and the RTT value (*timeAck*). It also applies a low-pass filter able to extract the low frequency components of the available bandwidth, since congestion occurs when the low frequency traffic rate exceeds the link capacity [5]. This method, shown in Alg. 2, is called every time an ACK is received.

After a congestion episode, the bandwidth estimation value (*w_bwe*) is used to establish the congestion window and the slow start threshold values. Congestion episodes are associated to the reception of three duplicate ACKs (*in receivedDuplicateAck*), or to a retransmission timeout exceeded event (in *processRexmitTimer*).

The *recalculateSlowStartThreshold* method calculates the slow start threshold using the estimated bandwidth and the minimum RTT (*RTTmin*). This minimum RTT is calculated each time a non duplicate ACK is received by the *receivedDataAck* method; in particular, it relies on a mechanism similar to the one described for TCP Vegas, where the *dataSent* method is redefined to register the sending time of each segment.

4. VALIDATION AND PERFORMANCE ASSESSMENT

In order to verify the correctness of the implementations described above, a series of simulations have been carried out to assess if the behavior obtained is correct according to previously publicated results [4, 5]. Additionally, we have compared the performance of the implemented protocols - TCP Vegas and TCP Westwood - with the performance of the existing implementation of TCP Reno in terms of throughput and retransmission behavior.

OMNeT++ version 4.2.2 was used for our experiments, where the target scenario defines two terminals, client and server, connected through a generic channel. The channel data rate is set to 100Mbps, and the values of delay and packet error rate (PER) are experiment-dependent in order to model different channel conditions.

The client requests a 10 MiB file to the server at the beginning of each simulation. The data transfer mode used is "stream of bytes", and a MSS (Maximum Segment Size) of 1460 bytes has been chosen to match typical operating system configurations.

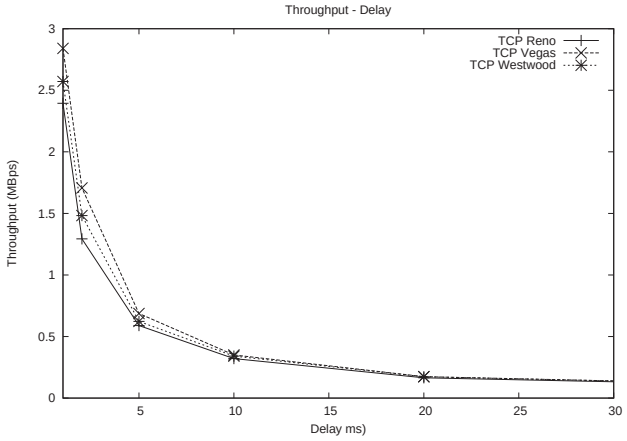

Figure 1: Throughput performance when varying delay (PER of 0.5%).

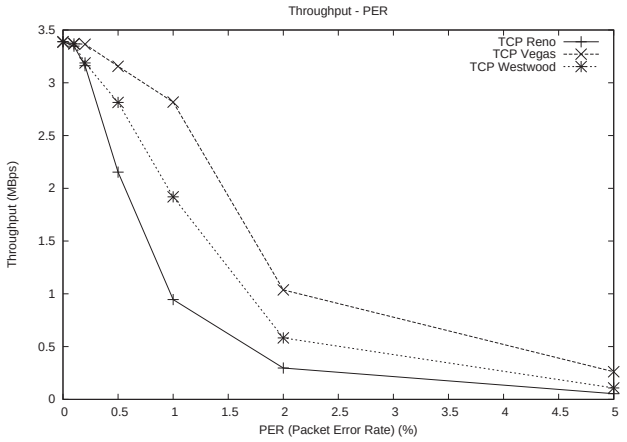

Figure 2: Throughput performance when varying the packet error rate (Delay of 1ms).

Simulations were performed taking 5 samples per configuration (<delay, PER, protocol> tuple), which are enough to guarantee a maximum margin of error equal to 10% for the mean value, with a degree of confidence of 90%.

Fig. 1 shows how the throughput of TCP Reno, TCP Vegas and TCP Westwood evolves when varying the delay value for a fixed PER of 0.5%. It can be noticed that the three protocols behave in a similar way. For small delay values, TCP Vegas and TCP Westwood present small performance improvements compared to TCP Reno. These differences become negligible for high delay values, being performance virtually the same for all three protocols.

Similarly, in Fig. 2 one can observe how the throughput of the three protocols varies when increasing the packet error rate in the channel (delay is fixed at 1ms). The graph shows a same overall trend for the three protocols with increasing PER, although TCP Vegas and TCP Westwood achieve significant performance improvements with respect to TCP Reno; in this experiment, TCP Vegas is the best performing protocol.

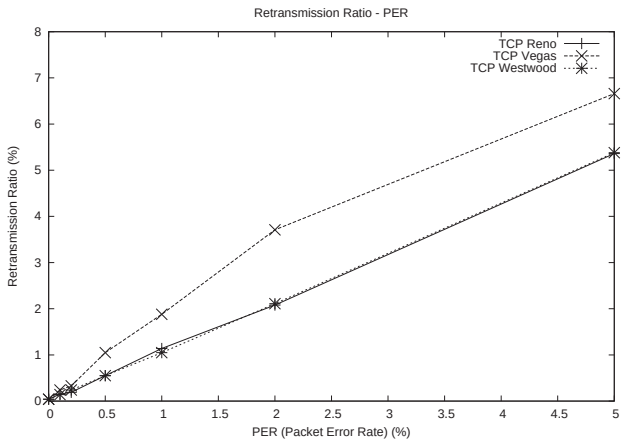

Figure 3: Retransmission ratio when varying the packet error rate (Delay of 1ms).

To obtain further insight about the different performance levels achieved, the retransmission ratio has also been measured; these results were again obtained by varying the PER, and defining a fixed delay value of 1ms. In Fig. 3 we can see that the retransmission ratio of TCP Reno and TCP Westwood is similar, although TCP Vegas presents a higher value. Since higher values are associated with more traffic overhead, TCP Vegas has a more negative impact on network bandwidth compared to the other protocols. This behavior of TCP Vegas is associated to its retransmission mechanism, which triggers packet retransmissions whenever the Vegas timeout is exceeded; the standard TCP behavior is that packet retransmissions are triggered by regular retransmission timeouts, or upon the reception of 3 duplicate ACKs.

Overall, the obtained results validate the implementations of the target protocols since the performance levels achieved fall within the margin of improvement stated in the original papers [4, 5]: about 46% of improvement over Reno for TCP Vegas, and about 30% of improvement over Reno for TCP Westwood. Also, we find that the behavior of the three protocols is quite similar for increasing PER and delay values, being performance differences significantly reduced for medium/large values of delay and packet loss.

5. CONCLUSIONS

The standard TCP protocol often offers a reduced performance in wireless environments, which is mostly due to the differences between these networks and the wired networks for which the protocol was created.

The work presented in this document focuses on the implementation and subsequent analysis of two existing alternatives to standard TCP - TCP Vegas and TCP Westwood - for the INET Framework of the OMNeT++ simulator. The goal is to provide a basic environment for researchers to develop new TCP implementations, mostly oriented to wireless networks, by offering a set of well-established reference protocols (Reno, Vegas and Westwood) according to current literature.

Experimental results show that TCP Vegas outperforms TCP Reno by up to 46% in terms of maximum throughput achieved, while TCP Westwood offers throughput improvements of up to 30% over Reno. These performance levels agree with previously publicated results.

As a future work, we believe it would be interesting to develop new TCP variants in order to promote the comparison of new protocols against the most effective solutions available in the literature for this research area.

Acknowledgments

This work was partially supported by the *Ministerio de Ciencia e Innovación*, Spain, under Grant TIN2011-27543-C03-01.

6. REFERENCES

[1] E. Ayanoglu, S. paul, T. F. LaPorta, K. K. Sabnani, R. D. Gitlin. “AIRMAIL: A Link-Layer protocol for Wireless Networks”, ACM ACM/Baltzer Wireless Networks Journal, vol. 1, pp. 47-60, February 1995.

[2] A. Bakre, B. R. Badrinath, “I-TCP: Indirect TCP for Mobile Hosts”, Proc. 15th International Conf. On Distributed Computing Systems (ICDCS), pp. 136-143, May 1995.

[3] H. Balakrishnan, S. Seshan, R. H. Katz, “Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks”, ACM Wireless Networks, 1(4), December 1995.

[4] L.S. Brakmo, S. W. O’Malley, L.L. Peterson, “TCP Vegas: New Techniques for Congestion detection and Avoidance”, Proc. ACM SIGCOMM, pp. 24-35, 1994.

[5] C. Casetti, M. Gerla, S. Mascolo, M.Y. Sanadidi, R. Wang, “TCP westwood: end-to-end congestion control for wired/wireless networks”, Journal of Wireless Networks, 8(5), pp. 467-479, September 2002.

[6] C. P. Fu, S. C. Liew, “TCP Veno: TCP Enhancement for Transmission over Wireless Access Networks”, IEEE JSAC, 21(2), pp. 216-228, February 2004.

[7] J. Kurose, K. Ross, “Computer Networking – A Top-Down Approach”, 4th ed., pp. 284, Addison Wesley, 2008.

[8] M. Mathis, J. Mahdavi, S. Floyd, A. Romanov, “TCP Selective Acknowledgment Options”, RFC 2018, October 1996.

[9] M. Mathis, J. Mahdavi, “Forward Acknowledgment: Refining TCP Congestion Control”, Proc. ACM SIGCOMM, pp. 281-291, 1996.

[10] J. Postel, USC/Information Sciences Institute, ”Transmission Control Protocol”, RFC 793, September 1981.

[11] D. Raychaudhuri, M. Gerla, “Emerging Wireless Technologies and the Future Mobile Internet”, Cambridge University Press, 2011.

[12] W. Richard Stevens, Gary. R. Wright, “TCP/IP Illustrated: The Implementation, Vol. 2.”, Pearson Education, 1995.