

# Progress in Various TCP Variants (February 2009)

B. Qureshi, M. Othman, *Member, IEEE*, and N. A. W. Hamid

**Abstract**—Transport Control Protocol (TCP), a basic communication language, consists of a set of rules that control communication. There are many versions of TCP which modified time to time as per need. Initially we discuss the basic functions of TCP and their role to control the congestion then graphically examine slow start, congestion avoidance, fast retransmission and fast recovery. This paper compares the performance of different TCP variants specifically Tahoe, Reno, New Reno, Westwood, Selective Acknowledgment (SACK), Forward Acknowledgment (FACK) and Vegas. TCP Vegas algorithm is explained with new structure mechanism and new congestion avoidance and modified slow start mechanisms. Subsequently, a table derived evaluates TCP variants on the basis of algorithm. We conversed the progress, and evaluated advantages and disadvantages of above TCP variants.

**Index Terms**—Congestion Avoidance, Fast recovery, Fast retransmission, Slow start

## I. INTRODUCTION

THIS overview enlightens different TCP versions algorithm like TCP Tahoe, Reno, New Reno, SACK, FACK and Vegas, on the basis of fundamental functions such as slow start, congestion avoidance, fast retransmission and fast recovery. Current TCP achievements contain several algorithms designed to run network congestion while maintaining throughput. The authors [1] argued and compared the performance of TCP Reno and TCP Vegas. This paper emphasize that due to the use of round trip times measurement, window dynamics of TCP Vegas are much more stable than those of TCP Reno, resulting in more efficient utilization of the network recourses. TCP Reno discriminates against users with long propagation delays; whereas TCP Vegas fairly shares the available bandwidth between users, whatever their propagation delays are. A paper [2] explains the congestion control algorithm in simulated implementation of SACK TCP and shows that the

selective acknowledgments are not required to solve Reno TCP's performance problems when multiple packets are dropped. The absence of selective acknowledgement does impose limits to TCP's ultimate performance. In particular, the paper shows that without selective acknowledgments, TCP implementations are constrained to either retransmit at most one dropped packet per round trip time, or to retransmit packets that might have already been successfully delivered. Forward Acknowledgment (FACK) congestion control algorithm developed [3], addresses many performance problems in the internet. FACK is designed on the basis of first principle of congestion control and TCP SACK option. It achieves more exact control over the data flow in the network when the congestion is decoupled from other algorithms like data recovery. In this paper two algorithms are designed to develop the performance in the exact state. Simulation is used for the comparison of FACK with Reno and Reno with SACK. Finally, it proves the impact on the potential performance of FACK in the internet.

## II. BASIC FUNCTIONS OF TCP

TCP is very complex but reliable, guaranteed and connection oriented transport protocol that provides a streaming service to its applications. When a TCP connection is established between sending and receiving processes, the sender writes a stream of bytes or characters into the connection and the receiver reads these bytes from the connection. TCP achieve reliability because it uses positive Acknowledgement (ACK) by receiver with retransmission of the packets. TCP implements variable sliding window size scheme (Advertised window) to accomplish the flow control and allow efficient utilization of network bandwidth. The receiver reads out the usable window size so that it can regulate the flow of data from the sender in a manner that does not overflow its buffers.

The sender is then able to transmit up to a full window's worth of packets before requiring an acknowledgment. To make efficient use of available bandwidth, the window size increased with the bandwidth of the connection and the delay in the path. The available bandwidth multiplied by the delay is referred to as the delay-bandwidth product.

Acknowledgments used by TCP are cumulative that provide the sender with sequence number of next packet that

Manuscript received October 19, 2008.

B. Qureshi is a PhD candidate in the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia. (phone: +603-89466535; fax: +603-89466576; e-mail: barkat662003@yahoo.com.)

M. Othman, is with the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia. (e-mail: mothman@fsktm.upm.edu.my).

N. A. W. Hamid is with the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia. (e-mail: asila@fsktm.upm.edu.my).

the receiver expects. ACKs are only sent in response to a packet being received rather than at particular intervals. Thus, if no packets are arriving, no ACKs will be sent. Cumulative acknowledgments are efficient in the sense that an ACK does not have to be sent for every packet received. They are also unclear because they do not explicitly inform the sender of any lost or damaged packets.

The time needed by a packet to travel from source to destination and from destination back to source is called the round-trip-time (RTT). The RTT is measured by TCP and used to calculate a value for the retransmission timer. The retransmission timer is set when a packet is transmitted and if a timeout occurs at the sender before an ACK for the packet is received, the packet is sent again. This feature ensures reliability because it allows TCP to detect losses and recover from them.

When heavy traffic slows down the network response time, this state is known as congestion. It is the main issue for TCP to resolve network congestion for which TCP implements a set of mechanisms collectively called congestion control, as described below:

#### A. Congestion Control

The basic role, to control congestion, is adjust the transmission window of the sender in such a way that buffer overflow is prevented at not only the receiver but also at the intermediate routers. To achieve this, TCP uses another window control variable called congestion window ( $cwnd$ ). TCP maintains congestion windows, which represent an estimate number of segments that can be injected in to network without causing congestion (segment is any TCP data or Acknowledgement packet or both).

The challenge is the utilization of available buffer space in network routers. Routers do not participate at the TCP layer and can not use TCP ACK segments to adjust the window.

To resolve this problem, TCP assumes network congestion whenever a retransmission timer expires, and it reacts to network congestion by adjusting congestion window using three algorithms, namely *slow start*, *congestion Avoidance* and *multiplicative decrease*.

#### B. Slow Start and Congestion Avoidance

The slow start state begins with the small window size that increases slowly as ACKs arrive, this is the rule behind the slow start mechanism. The initial value of  $cwnd$  is set between 1 to 4 packets in the beginning of this state. Receiver maintains an advertise window ( $rwnd$ ) which indicates the maximum number of bytes it can accept. The value of the  $rwnd$  is sent back to the sender collectively with each packet going back. The amount of outstanding data  $wnd$  is limited by the minimum of  $cwnd$  and  $rwnd$ . New packets are only sent if allowed by both congestion window and receivers advertised window, as follows:

$$wnd = \min(rwnd, cwnd) \quad (1)$$

In slow start phase, when ACK received the congestion window is increased by 1 segment ( $cwnd = cwnd + 1$ ). This phase is used at the times when new connections are

established and after the retransmission due to time-outs occurring. Slow start increases  $cwnd$  exponentially by adding one packet each time it receives an ACK at sender point. Slow start control the window size controls until  $cwnd$  achieves a threshold called slow start threshold ( $ssthresh$ ). When  $cwnd$  reaches  $ssthresh$ , the congestion avoidance state begins. In congestion avoidance phase the congestion window is increased by one packet per round trip time, at this state window increased linearly. When non duplicate ACK received  $cwnd$  is increased as follows:

$$cwnd = cwnd + MSS * MSS / cwnd \quad (2)$$

Equation (2) provides an acceptable approximation to the underlying principle of increasing  $cwnd$  by 1 full sized segment per RTT [4]. When time out occurs, the  $ssthresh$  is reduced to one-half the current window size as follows:

$$ssthresh = \min(rwnd, cwnd) / 2 \quad (3)$$

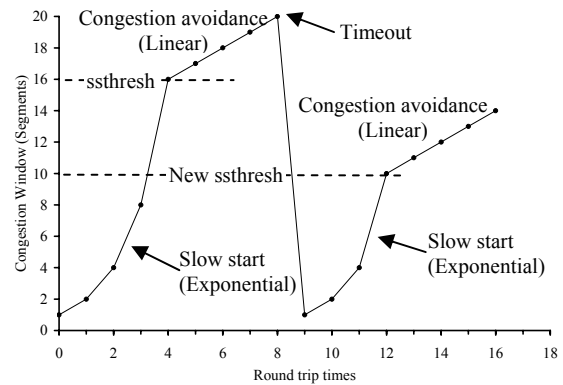


Fig. 1. TCP Slow start and congestion avoidance phase

The Fig. 1 shows changes in congestion window throughout the slow start and the congestion avoidance phase. The illustration explains that the position of  $ssthresh$  is 16 at preliminary state and timeout happens after 8 round trip times. At that time, the  $cwnd$  assumes a value of 20 therefore the position of new threshold after timeout (new  $ssthresh$ ) is at 10.

#### C. Fast Retransmit and Fast Recovery

The function of fast retransmit and fast recovery algorithm shown in Fig. 2, is assigning TCP to detect loss before the transmission timer expires. When out-of-order packet arrives in the receiver, the receiver transmits duplicate ACK. The sender obtains it as a packet loss or packet delay. Afterward if three or more duplicate ACKs are received in a row, the sender concludes that the misplaced packet was lost. Now the sender performs retransmission of what happens to be the misplaced packet, without waiting for a coarse-grain timer to expire. The  $ssthresh$  is set to the same value as given in the case of timeout Equation (3). Subsequent to *fast retransmission*, *fast recovery* is performed until all the data is convalesced. Consequently, the congestion window is set to three packets more than  $ssthresh$ . These supplementary three packets take account of the number of packets (three) that have left the network and which the receiver has buffered.

Therefore, TCP is capable of evading the needless slow starts due to unimportant congestion occurrence, with fast retransmission and fast recovery.

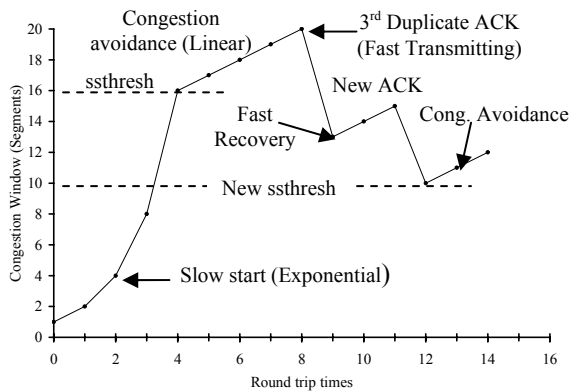


Fig. 2. TCP Slow start and congestion avoidance phase

### III. PERFORMANCE EVOLUTION OF TCP VARIANTS

The most pervasive transport protocol engaged is the Transmission Control Protocol.

In the very earliest achievement of TCP, little was done to minimize the network congestion. Implementation used cumulative positive acknowledgements and the expiry of a retransmit timer to provide reliability based on a simple go-back-n model. Several succeeding versions of TCP based on congestion control and avoidance mechanism have been developed, nowadays. In this section, we argue the performance of various TCP versions like regarding Tahoe, Reno, New Reno, SACK, FACK and Vegas.

#### A. TCP Tahoe

One of the TCP congestion control algorithms, is TCP Tahoe described [5], adds some new and enhance the earlier TCP implementation, including *slow start*, *congestion avoidance* and *fast retransmission*. This enhancement comprises change in round-trip-time estimation used to position retransmission time out values. TCP Tahoe fast retransmission algorithm outperforms the most when the packets are lost due to congestion. Sender should be waiting for retransmission timer to expire in the without fast retransmit algorithm. Hence, fast retransmission can save numerous seconds every time packet loss occurs, and the throughput is improved, consequently.

The shortcoming in TCP Tahoe is that packet loss is detected after the whole timeout interval. When a packet loss is detected, TCP Tahoe performance becomes slow. Due to this reason transmission flow decreases rapidly.

While fast retransmit makes Tahoe perform significantly better than a TCP implementation in which means of loss detection are merely the retransmission timers. It obtains significantly less than optimal performance on high delay-bandwidth connections because of its initiation of slow start (which TCP Reno conversed in following section). Also, in the case of multiple losses within a single window, it is

possible that the sender will retransmit packets which have already been delivered [2].

#### B. TCP Reno

The Reno TCP mechanism is similar to the TCP Tahoe except it maintains improvements over Tahoe by adding to the *fast recovery* phase known as *fast recovery* algorithm [6].

The significant improvement in TCP Reno in contrast to TCP Tahoe, prevent the communication path “*pipe*” from going empty after fast retransmit, and in that way it avoids slow start to fill it again after a packet loss.

TCP Reno maintains the clocking of new data with duplicate ACKs which make it more beneficial than TCP Tahoe. In this way, TCP allows to directly cut its throughput in half without the need for a slow start period to reestablish clocking between the data and ACKs. This improvement has the most noticeable effect on long delay-bandwidth connections where the slow start period lasts longer and large windows are needed to achieve optimal throughput.

When a single packet is lost from a window of data, TCP Reno maintains it by fast recovery mechanism, in contrast when multiple packets are lost, Reno’s performance are same here as Tahoe. This indicates that if multiple packets are lost from the same window, TCP Reno almost immediately drag out of fast recovery, and stop until no new packet can be sent.

The above discussion leads to conclusion that fast recovery mechanism introduced by TCP Reno handles multiple packet losses within a single window poorly.

#### C. TCP New Reno

Hoe [7] intended that the experimental version of TCP Reno is known as TCP New Reno. It is slightly different than TCP Reno in fast recovery algorithm. New Reno is more competent than Reno when multiple packets losses occur. New Reno and Reno [8] both correspond to go through in to fast retransmit when multiple duplicate packets received, except another way that afterward it does not come out from fast recovery phase until all outstanding data at the time it entered fast recovery are acknowledged. It implies that in New Reno partial ACK do not take TCP out of fast recovery but they are treated as an indicator that the packet immediately following the acknowledged packet in the sequence space has been lost, and should be retransmitted. Therefore, when multiple packets are lost from a single window of data, at this time New Reno can improve without retransmission time out. The retransmitting rate is one loss packet per round trip time until all of the lost packets from that window have been transmitted. It exist fast recovery when all the data is injected into network, and still waiting for an acknowledgement at the movement that fast recovery was initiated, has acknowledged.

The critical issue in TCP New Reno is that it is capable of handling multiple packet losses in a single window. It is limited to detecting and resending at most one lost packet per round-trip-time. This insufficiency becomes more distinct as the delay-bandwidth becomes greater. More importantly, there are situations where stalls can still occur if packets are lost in

successive windows. Also, like all of the previous versions of TCP discussed above, New Reno still infers that all lost packets are caused by congestion and it may therefore unnecessarily cut the congestion window size when errors occur.

#### D. TCP Westwood

The modified version TCP Reno is TCP Westwood, [9] it extends the window control and backoff processing. When the packet loss occurs, three DUPACKs are received; the sender uses the bandwidth estimates to properly adjust the congestion window and the slow start threshold. 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 Westwood avoids reductions of *cwnd* and *ssthresh* that can be excessive or insufficient. Westwood adopt a strategy of Additive Increase and Adaptive Decrease (AIAD) instead of Additive Increase and Multiplicative Decrease (AIMD). Therefore, TCP Westwood guarantees both the faster recovery and more effective congestion avoidance.

#### E. TCP SACK

It is an enhanced version (E V) of TCP Reno and New Reno with Selective Acknowledgement. The two main problems, the detection of multiple lost packets and the retransmissions of more than one lost packet per round trip time, can be solved through TCP SACK. In this protocol, the packets are acknowledged selectively rather than cumulatively. This application allows a TCP receiver to send SACK information as a set of options within the TCP header which is complimentary to the existing TCP ACK [10]. The new option fields indicate the starting and ending sequence of non-contiguous sets of data existing at the receiver. Depending on the other TCP options being used by the connection, a maximum of either two or three blocks of data may be reported by a single ACK. Receivers include SACK information in the TCP header only when duplicate ACKs are sent in response to the arrival of an out-of-order packet. A new scoreboard matrix is introduced at the sender to keep track of this SACK data, and a new pipe mechanism is used to track the number of packets currently in transit (i.e. in the “network data pipe”). SACK performs fast retransmit just like New Reno. It enters the fast retransmit phase when a loss is detected, and it exits when all of the data has been acknowledged which was outstanding when the fast retransmit phase began.

SACK still makes no attempt to distinguish between losses due to congestion and errors on the wireless link (it just decreases the effects losses have on performance). Also, it seems that with the currently proposed implementation of SACK, there are still situations where stalls could occur if packets are lost in very specific patterns.

#### F. TCP FACK

The development in TCP SACK with Forward Acknowledgement is identified as TCP FACK. The utilization

of TCP FACK is almost identical to SACK but it establishes a little enhancement evaluated to it. It uses SACK option to better estimate the amount of data in transit [3].

TCP FACK introduces a better way to halve the window when congestion is detected. When *cwnd* is immediately halved, the sender stops transmitting for a while and then resumes when enough data has left the network. This unequal distribution of segments over one RTT can be avoided when the window is gradually decreased [3].

When congestion occurs, the window should be halved according to the multiplicative decrease of the correct *cwnd*. Since the sender identifies congestion at least one RTT after it happened, if during that RTT it was in slow start mode, then the current *cwnd* will be almost double than *cwnd* when congestion occurred. Therefore, in this case, *cwnd* is first halved to estimate the correct *cwnd* that should be further decreased.

#### G. TCP Vegas

Vegas algorithm predicts the beginning of congestion by observing the difference between the expected rate and actual rate [11]. Vegas adjust congestion window at source sending rate in an effort to keep a small number of packets buffered in the routers along the transmission path. TCP Vegas sender stores the current value of the system clock for each segment it sends. Therefore, it is able to know the exact RTT for each sent packet. TCP Vegas established the following latest developments.

(i) *New Retransmission Mechanism*: In this algorithm duplicate acknowledgement is received by the sender. It checks if (current time – segment transmission time) > RTT. If this statement is true, the sender provides a retransmission waiting neither for the traditional retransmission timeout nor for three duplicate ACKs. This development can prevent a state in which the sender receives three duplicate ACKs and must rely on the coarse-grain timeout.

Vegas gives individual consideration to the first two partial ACK after a retransmission to avoid multiple packet loss problems [12]. The sender decides it as a multiple packet loss by checking the timeout of unacknowledged packets. If any timeout occurs, the sender immediately retransmits the packet without waiting for any duplicate ACK. Also to avoid the over-reduction of window size due to the loss occurred at the previous window size, Vegas compares the timestamp of the retransmitted packet and the timestamp of the last window decrease. When the retransmitted packet is sent before the last decrease, Vegas will not decrease *cwnd* on receiving duplicate ACKs for this packet because this packet loss occurred due to the previous window size. For Vegas, it is very important that designing such a mechanism to avoid unnecessary reduction of *cwnd* because of its quick response of packet loss.

(ii) *New congestion avoidance*: When receiving an ACK, the sender calculates the difference of the expected and the actual throughputs as follows:

$$\begin{aligned} \text{diff} &= (\text{expected} - \text{actual}) * \text{baseRTT}, \\ \text{expected} &= \text{cwnd}/\text{baseRTT}, \end{aligned}$$

actual =  $cwnd$  / average measured RTT

Expected throughput represents the available bandwidth for this connection without network congestion, and actual throughput represents the bandwidth currently used by the connection. Vegas defines two thresholds ( $\alpha$ ,  $\beta$ ) as a tolerance that allows the source to control the difference between expected and actual throughputs in one RTT. The  $cwnd$  is increased by one packet if  $\text{diff} < \alpha$  and decreased by one packet if  $\text{diff} > \beta$ . That is

$$cwnd = cwnd + 1, \text{ if } \text{diff} < \alpha$$

$$cwnd = cwnd - 1, \text{ if } \text{diff} > \beta$$

$$cwnd = cwnd, \text{ otherwise}$$

(iii) *Modified slow start mechanism*: To detect and avoid congestion during slow start, Vegas doubles its window sizes every other RTT, not every RTT as does Reno. The cause for this alteration is that when a connection starts at the initial state there is not any initiative for the sender about the existing bandwidth. In consequence, it possibly will happen that throughout exponential increase it over shoots the existing bandwidth by a large amount as well as congestion. When the edge value is achieved in the difference between current RTT and the last RTT at this state slow start is ended. This represents a modification compared to others TCP versions where the boundary is set in  $cwnd$  size. The Table 1 expresses the evaluation of various TCP variants, on the basis of algorithms.

TABLE 1

TCP VARIANTS EVALUATION ON THE BASIS OF ALGORITHMS

| Algorithms/<br>TCP<br>Variants     | TCP<br>Tahoe | TCP<br>Reno | TCP<br>New<br>Reno | TCP<br>Westwood | TCP<br>SACK | TCP<br>FACK | TCP<br>Vegas |
|------------------------------------|--------------|-------------|--------------------|-----------------|-------------|-------------|--------------|
| Slow Start                         | Yes          | Yes         | Yes                | Yes             | Yes         | E V         | E V          |
| Congestion<br>Avoidance            | Yes          | Yes         | Yes                | Yes             | Yes         | Yes         | E V          |
| Fast<br>Retransmit                 | Yes          | Yes         | Yes                | Yes             | Yes         | Yes         | Yes          |
| Fast<br>Recovery                   | No           | Yes         | E V                | E V             | E V         | E V         | Yes          |
| Retransmissi<br>on<br>mechanism    | N            | N           | N                  | N               | N           | N           | N M          |
| Congestion<br>Control<br>mechanism | N            | N           | N                  | N               | N           | N M         | N M          |
| Selective<br>ACK<br>mechanism      | No           | No          | No                 | Yes             | Yes         | Yes         | No           |

(N = Normal, E V = Enhanced Version, N M = New Mechanism)

#### IV. CONCLUSION

In this paper we conclude that congestion is the main issue in different variants of TCP. If throughput is increased the congestion will be reduced. Tahoe performance becomes slow when a packet loss occurs. Reno also behaves poorly like Tahoe when multiple packets loss occurs within window. New Reno is limited to detecting and resending at most one lost packet per round trip time. The new approach AIAD implemented by Westwood make it guaranteed for fast

recovery and efficient congestion avoidance. We discovered that there are fundamental restrictions imposed by the lake of selective acknowledgment in TCP. We examined a TCP implementation that incorporates selective acknowledgement SACK in to Reno TCP while making minimal changes to TCP's underlying congestion control algorithms. FACK introduces better way to halve the window when congestion is detected. In TCP Vegas, we focused on the performance such as average throughput and the average buffer occupation. Finally it is concluded that TCP Vegas, the window mechanism of which consists of stabilizing the window size to the optimal value plus a number of extra packets comprised between  $\alpha$  and  $\beta$ , is much more stable, efficient and fair than the other TCP Variants. We assume that TCP Vegas will open the way to further development of the TCP protocol.

#### REFERENCES

- [1] T. Bonald, "Comparison of TCP Reno and TCP Vegas: efficiency and fairness", in Performance Evaluation, Vol. 36-37, pp. 307-332, 1999.
- [2] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP", in ACM Computer Communication Review, Vol 26, pp. 5-12, 1996.
- [3] M. Mathis and J. Mahdavi, "Forward acknowledgment: refining TCP congestion control" in Proceedings of ACM SIGCOMM, pp. 181-191, 1996.
- [4] M. Allman, V. Paxson and W. R. Stevens, "TCP congestion control", in IETF RFC, 2581, 1999.
- [5] V. Jacobson, "Modified TCP congestion avoidance algorithm. end2end-interest mailing list", in Tech. Report in IEICE Transactions on Communications, 2007, E90-B(3):516-52, 1990.
- [6] J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP", in Proceedings of SIGCOMM Symposium, pp. 270-280, 1996.
- [7] V. Jacobson, "Congestion avoidance and control", in SIGCOMM Symposium on Communications Architectures and Protocols, pp. 314-329, 1999.
- [8] S. Floyd, T. Henderson, and A. Gurtov, "The new Reno modification to TCP's fast recovery algorithm, IETF RFC, 3782, 2004.
- [9] C. Casetti, M. Gerla, S. Mascolo, M. Y. Sanadidi and R. Wang, TCP Westwood: bandwidth estimation for enhanced transport over wireless link", in Wireless Network, Vol. 8, pp. 467-479, 2002.
- [10] M. Mathis, S. Floyd and A. Romanow, "TCP selective acknowledgment options", IETF RFC, 2018, 1996.
- [11] L. Brakmo, S.O. Malley and L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", in Proceedings of ACM SIGCOMM Symposium, pp. 24-35, 1994.
- [12] Y. -C. Lai and C.-L. Yao, "Performance comparison between TCP Reno and TCP Vegas". Computer Communication, 25: 1765-1773. 2002.



**B. Qureshi** received the B.Sc. and M.Sc. degrees in Mathematics and Computer Science from University of Sindh, Jamshoro, Pakistan, in 1989 and 1992, respectively. From January 1992 to November 1995, he was a System Analyst at Sindh Development Studies Centre (SDSC), University of Sindh. In November 1995, he joined Sindh Agriculture University Tandojam, Pakistan, as a Assistant Professor. He is now on study leave for pursuing Ph.D in Computer Networks, Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, University Putra Malaysia. His research work is in the area of Flow Control on Parallel TCP.



**M. Othman** is a Associate Professor at the Department of Communication Technology and Network, and Deputy Director of the InfoComm Development and Communication Centre, University Putra Malaysia. He received his PhD from the National University of Malaysia with distinction (Best PhD Thesis in 2000 awarded by Sime Darby Malaysia and Malaysian Mathematical Science Society). In 2002 till 2007, he received gold medal awards for Research and Development

Exhibition. His main research interests are in the fields of parallel and distributed algorithms, high-speed computer network, network management (security, wireless and traffic monitoring), grid computing, and scientific computing. He is a member of IEEE Computer Society, IEICE Communication and Engineering Science Societies, Malaysian National Computer Confederation and Malaysian Mathematical Society. He is also associated editor of *Pertanika Journal of Science and Technology* and *Malaysia Mathematical Science Journal*. He already published more than one hundred National and International journal papers and more than three hundred conference papers. He is also an associate researcher and coordinator of High Speed Machine at the Laboratory of Computational Science and Informatics, Institute of Mathematical Science (INSPEM), University Putra Malaysia.



**N. A. W Hamid** is a Lecturer at the Department of Communication Technology and Network, University Putra Malaysia. She received her PhD from University of Adelaide, Australia. Her main research interests are in the fields of parallel and distributed algorithms, high performance computer and grid computing. She is also an associate researcher of High Performance Computer at the South Australian Partnership of Advanced Computing (SAPAC) at University of Adelaide, Australia and Laboratory of Computational Science

and Informatics, Institute of Mathematical Science (INSPEM), University Putra Malaysia.