CHAPTER TWO

LITERATURE REVIEW

Nowadays, wireless technologies are almost everywhere. This extensive development of new technologies has motivated a large spectrum of networking researchers to enhance the communication system infrastructure standards for both hardware and software. The Transmission Control Protocol (TCP) is the most widely used transport protocol in today's networks. TCP achieves a seemingly impossible task: it uses the unreliable Internet Protocol datagram service to provide a reliable data delivery service. The standard TCP implementations TCP Reno and NewReno are the most common TCP implementations in the Internet. The standard congestion control mechanism of both implementations was designed according to wired links assumptions. From this, packet losses seldom occur and are mostly a signal of congested links. On the other hand, over wireless links, packets losses commonly occur due to a loss of transmission medium. Applying the same congestion control mechanism over wireless links results in a serious performance degradation in the communication system.

This chapter will present the background to common transport protocols in communication systems. The first section presents a brief introduction to layering concepts. TCP fundamentals are discussed in section two, while in section three we will present a detailed study of the congestion control mechanism of common TCP variants. Section four presents the literature on TCP congestion control mechanisms

over wireless networks. In section five, we introduce the Simple Network Transport Protocol (SCTP) protocol, and finally section six concludes chapter two.

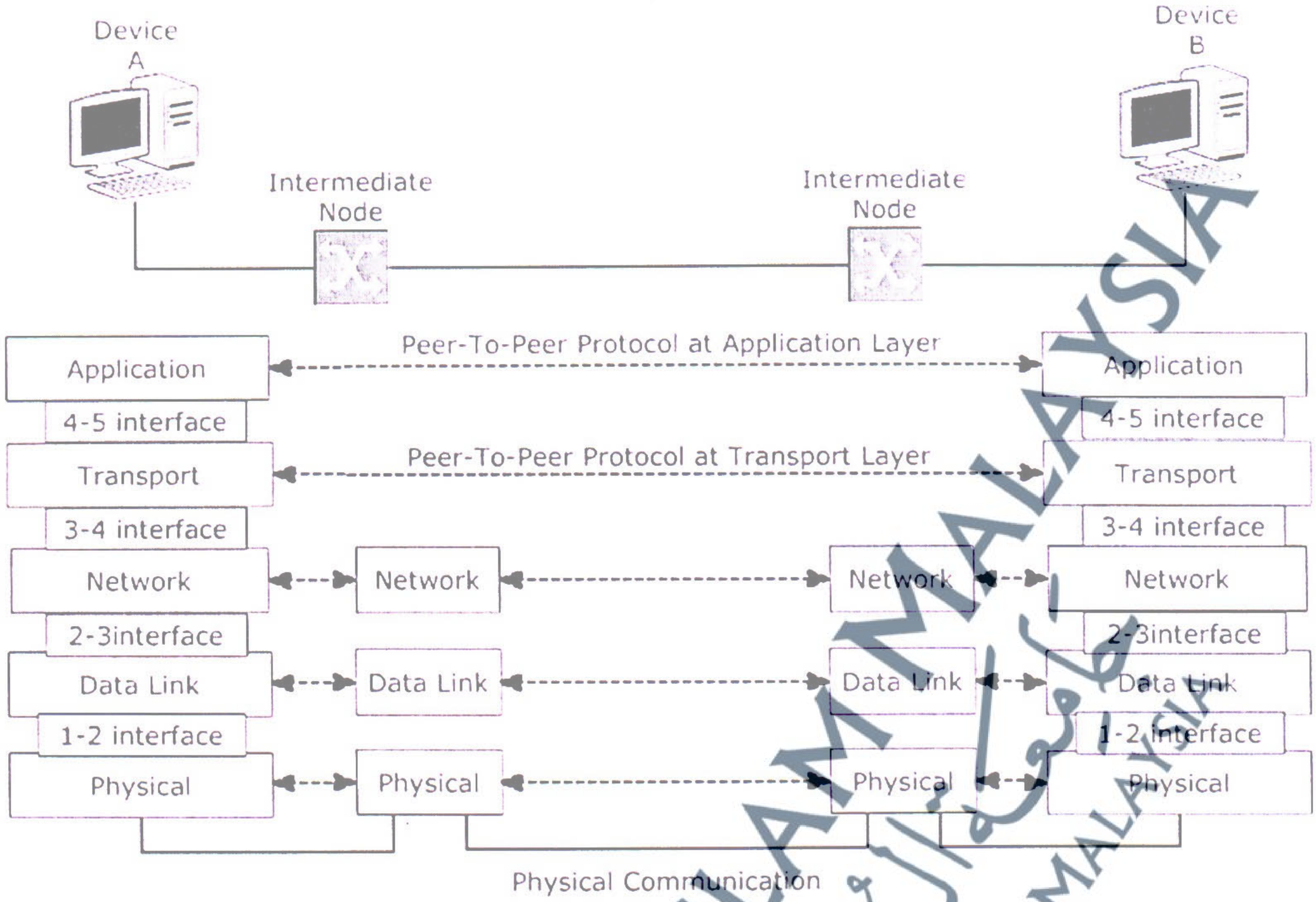
2.1 Layering concept

Computer networks are developed and designed by different entities. These heterogeneous networks can communicate with one another using the networks' standards. The ISO-OSI model and the Internet Model are currently the best known standards. The OSI model was proposed as the standard model for all communication networks from 1970 until 1990. After that, it was replaced by the TCP/IP or Internet Model as the standard framework used to design network systems (Peterson & Davie, 2007).

The International Standards Organization (ISO) was established in 1947. It is a multinational organization dedicated to worldwide agreement on standards. The ISO introduced the network communication Open System Interconnection (OSI) model in 1970. The OSI model is not a protocol; rather, it is a framework for understanding and designing heterogeneous networks to be flexible, robust, and interoperable. The OSI model is layered into seven separate but related layers including application, presentation, session, transport, network, link, and physical. Each layer is a part of the process of sending and receiving data between networks nodes (Forouzan & Fegan, 2007).

On the other hand, the Internet Model or the TCP/IP protocol stack is modelled into five layers: from the top, there are the application, transport, network, and data link layers, and at the bottom there is the physical layer, as described in Figure 2.1.

Figure 2.1: Internet Model (Adopted from Forouzan & Fegan, 2007)



Each layer has a series of protocols to provide the services needed from the top layer through the service interfaces between two sequencing layers. The application layer in the TCP/IP model is equivalent to the combined session, presentation, and application layers in the OSI model. This layer runs a range of application protocols (FTP, HTTP, Telnet, SNMP, and etc.) to provide interoperation to the end-user applications. The transport layer contains three main protocols: the Transmission Control Protocol (TCP), the User Datagram Protocol (UDP), and the Stream Control Transmission Protocol (SCTP). This layer is responsible for delivering data between two processes (running programs) through a process-to-process connection. The following section will discuss this layer in more detail.

The network layer runs a single protocol, the Internet Protocol. The IP supports the interconnection of multiple networking technologies into a single logical internetwork (Peterson & Davie, 2007). Other protocols that support the data

movement in this layer include the Address Resolution Protocol (ARP), the Reverse Address Resolution Protocol (RARP), the Internet Control Message Protocol (ICMP), and the Internet Group Message Protocol (IGMP).

The physical and data link layers in the TCP/IP model do not define any specific protocol; instead, these two layers support all the standard and proprietary protocols. These protocols are implemented by a combination of hardware and software specifications (e.g. a network adapter, or a network device driver).

2.1.1 Transport Layer

As discussed in the previous section, the transport layer in the TCP/IP model has three different protocols: UDP, TCP, and SCTP. Even though these protocols share the same function of providing a process-to-process connection, they use different mechanisms and have different capabilities (Chellaprabha, 2012). The form of extending the host-to-host delivery service of the network layer into a process-to-process communication service is the simplest possible transport protocol, the UDP.

The UDP is a connectionless transport protocol. The connectionless service of the UDP means that there is no relation between the user datagram (UPD packets) even if they are from the same client and transmitted to the same server. Furthermore, UDP's user datagrams are not numbered, there is no connection establishment and termination, and the user datagrams can be transmitted using different paths. Therefore, UDP is suitable for transmitting streaming video or multimedia, and it is recommended for those processes which require the sending of short messages (Peterson & Davie, 2007).

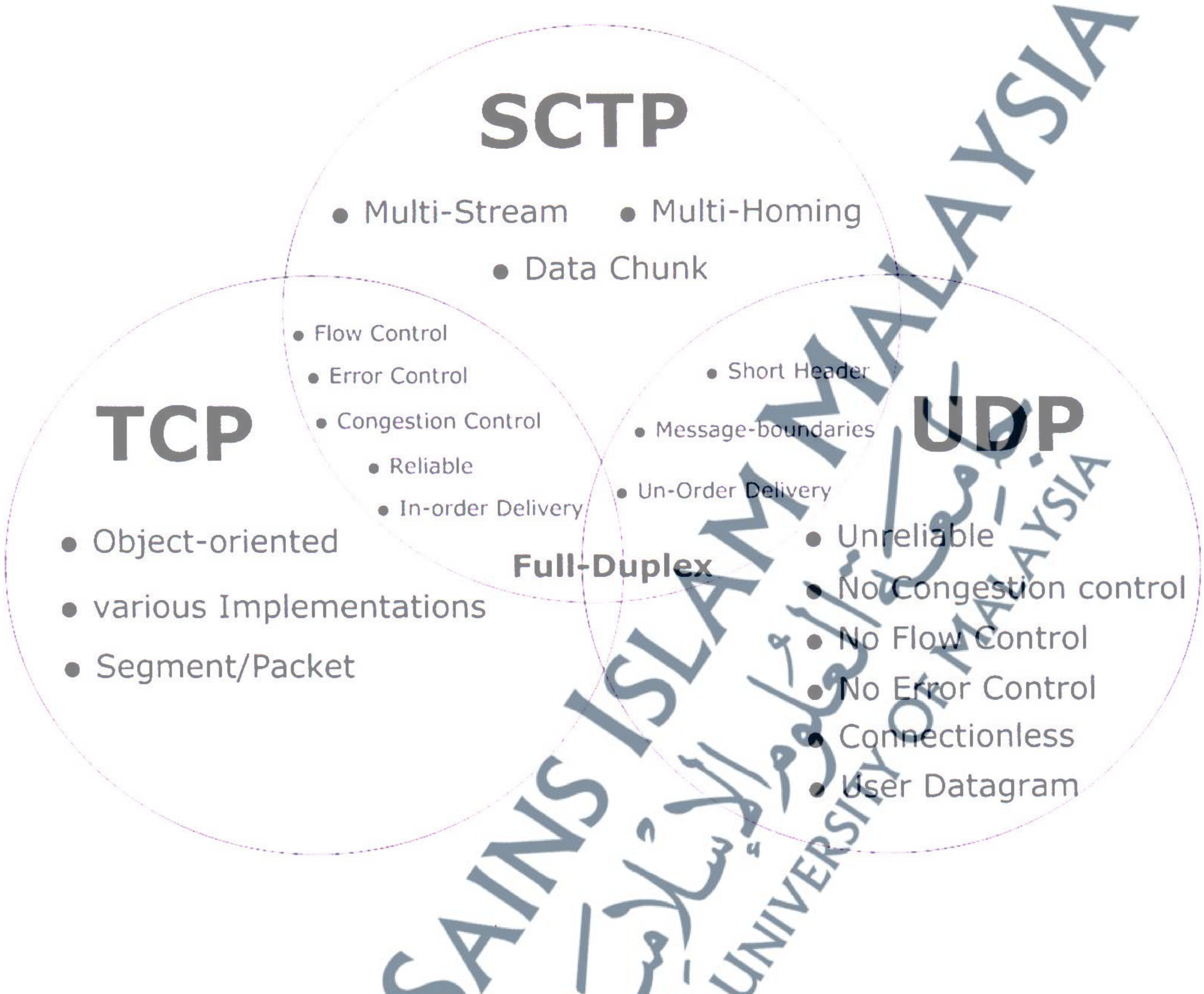
Moreover, UDP is an unreliable transport protocol, as it does not provide any form of control, either of flow, congestion, or error (Kozierok, 2005). However, UDP uses a checksum error mechanism to silently discard datagrams with checksum errors at the receiver side. Therefore, the flow and error control should be provided by the sender and the receiver sides processes. According to Kozierok (2005), UDP's strengths include the fact that it uses a light weight user datagram header, does not require an acknowledgment mechanism, and requires a very short processing time at the end nodes.

In contrast to the UDP, a more sophisticated transport protocol that provides reliable, connection-oriented, and byte-stream services is the TCP, which is widely used in today's networks. TCP guarantees reliable and in-order delivery of a stream of bytes (Peterson & Davie, 2007). It supports a full-duplex connection in pair byte streams using different flows in both directions. Moreover, it provides a flow control to limit the amount of data the sender can transmit based on the receiver limit. Also, TCP provides a highly tuned congestion control mechanism to keep the sender from overloading the network. Sections 2.3 and 2.4 will discuss the TCP protocol in more detail.

A newer reliable transport protocol was introduced in Stewart et al. (2000), known as the SCTP. SCTP was designed to transport Packet Switch Telephone Network (PSTN) messages over IP networks and it combines both UDP and TCP features. In addition, SCTP supports multi-streams and multi-homing connection services. SCTP is inherent in most TCP congestion control mechanisms, and also it provides both error and flow control. The comparative diagram in Figure 2.2 shows

the similarities and the differences between the three transport protocols: TCP, SCTP, and UDP.

Figure 2.2: Transport layer protocols



Even though the SCTP shares the best features of TCP and UDP, it has not replaced the famous TCP (Stewart & Amer, 2007). Currently, TCP NewReno is the standard transport protocol in the Internet Model. Therefore, this study focuses on enhancing the current transport protocol congestion control mechanism, the TCP NewReno, over the latest network technologies. We believe that the new mechanisms could be implemented within the SCTP congestion control as well.

2.2 Transmission Control Protocol (TCP)

The first outline of the Transmission Control Protocol (TCP) was given in Vinton & Kahn (1974). They introduced a protocol to support resource sharing in different packet switching networks. The main functions of the proposed protocol were

transmission failure handling, sequencing, flow control, end-to-end error checking, and process-to-process connection. In 1983 the ARPAnet (Advanced Research Projects Agency Network) replaced the Networks Control Protocol (NCP) by TCP in combination with the Internet Protocol (IP). Since that time, the TCP/IP has governed the data transmission over packet switching networks.

Over the years, many TCP variants have been proposed to support the many new networking technologies. Basically, TCP achieves a seemingly impossible task. It uses the unreliable IP protocol to provide a reliable data delivery service. Therefore, TCP has developed many mechanisms to ensure its reliability includes:

1. Sequencing mechanism to handle duplicates and out of order packets. Therefore, TCP at the sender side attaches a sequence number with each sent packet, and thus the receiver side can check in-order packet delivery, and avoid duplicate packets before delivering the coming packets to the applications.
2. A retransmission mechanism to handle lost and damaged packets. The TCP sender side starts a timer each time it sends a packet and waits for the receiver's acknowledgment. The TCP receiver side sends a positive *ACK* to the sender side each time it has received a correct packet. If the packet's timer expires before receiving its *ACK*, the TCP sender side must retransmit the same packet, again assuming that it has been lost or damaged.
3. Mechanisms to avoid reply. Long delay could cause replay errors, where a packet from an earlier connection could be arrived in a later connection and accepted with its sequence number. Later, the correct packet with the same sequence number will be discarded when it arrives as a duplicate packet.

Thus, the TCP attaches a connection session ID with each packet and makes sure this ID will be expired within a reasonable period.

The following three sections describe other TCP mechanisms in more detail, including Round Trip Time and Timeout, TCP flow control, and TCP congestion control.

2.2.1 TCP Round Trip Time and Timeout

The TCP uses timers to handle damaged and lost packets in the absence of the receiver's acknowledgment. The TCP at the sender side sets a timer when it sends a packet, and if this timer expires before receiving a packet's ACK, the sender assumes a packet is lost and automatically retransmits the same packet. This timer is called the retransmission timeout (*RTO*). A critical element to preserve better TCP performance and good throughput is how to calculate the *RTO* intervals and how frequently timeout occurs.

The fundamental point here to set proper values of *RTO* is measuring the round trip time (*RTT*) values in a given connection. As mentioned before, *RTT* is the time between sending a packet and receiving its acknowledgment. The TCP protocol specification (Postel, 1981) suggests estimating the main *RTT* using a low-pass filter, according to the following equation:

$$R \leftarrow \alpha R + (1-\alpha)M \tag{2.1}$$

Where R is the estimated *RTT*, M is last measured *RTT* and α is a filter gain constant and equal to 0.9. When R is updated, then the *RTO* for the next sent packet is set to $R=\beta$, where β is the delay variance factor with a recommended value of $\beta=2$. However, Jacobson 1988 claimed that this approach cannot cope well with large *RTT*

fluctuations because it causes unnecessary retransmission. These unnecessary retransmissions add more traffic to a loaded network. Therefore, Jacobson suggested using both the variance in *RTT* and the smoothed *RTT*. In other words, calculating the *RTO* based on both the mean and the variance of *RTT* is as described in Jacobson (1988) according to the following equations:

$$RTO = A + 2D \tag{2.2}$$

Where A is the smoothed *RTT* and D is the *RTT* mean deviation. After further research, Jacobson used $4D$ to calculate *RTO* (1990).

Karn and Partridge (1987) argue that we cannot update the *RTT* estimator when timeout occurring because we cannot judge which transmission the *ACK* corresponds to (either the packet was delayed in reaching its destination or its *ACK* was delayed). Therefore, they do not calculate a new *RTO* until they receive an *ACK* from a non-retransmitted packet. This is known as Karn's algorithm.

2.2.2 TCP Flow Control

One of the main features of the TCP is matching the transmission rate of the sender to the receiving capacity at the destination. At the same time, the transmission rate should be high enough to ensure good performance. Therefore, the TCP uses a 16-bit field at the TCP's header to advertise the receiver's capacity. Since the receiver window is bounded by 16-bits, this provides a maximum window size of 65,535 bytes.

The simplest form of flow control is known as Stop-and-Go, where the sender side waits for the receiver's *ACKs* before sending new packets. However, the stop-and-go technique provides reliable connection but results in extremely low performance. Therefore, TCP uses the sliding window technique to maximize network

utilization. With the sliding window, the window size advertises to the sender how much data there is, starting from the current position of the sending stream being sent without waiting for the receiver's *ACKs*. As the receiver's *ACKs* arrive, the window slides forward to cover new data in the byte stream. The following figure describes the "sliding window" concept:

Figure 2.3: The sliding window technique

As shown above, the sender can send the data stream within the window boundary without waiting for the receiver's *ACKs*. Packets 1, 2, and 3 have been sent and *ACKed*, and the bytes sent ahead to the receiver window are waiting for the window to slide forward before they can be transmitted. The receiver adjusts the receiver window each time it sends *ACK* to the sender side. Thus, the transmission rate is limited by the receiver's ability to accept and process new packets. For example, a source and a sink connected with link speed of 1Gbps (i.e. 125,000,000 bytes can be transmitted per a second). If the receiver's window is 16bits field (i.e. 65,535 bytes) this means, at best, 65,535/125,000,000 or 0.0005 of the available bandwidth will be used. Therefore, the Internet Engineering Task Force releases a

"Window scale option" standard in RFC1323. According to this standard, the receiver can increase its window up to 32 bits (i.e. 4 billion bytes).

2.2.3 TCP Congestion Control

The massive growth in the Internet and its applications has seen the emergence of the concept of TCP congestion and control. In 1980, the first series of "Internet congestion-collapse" happened (Mahmoodi, 2009). The TCP plays a major role in controlling and avoiding network congestion, mainly by reducing the sending rate according to the network status. The main idea of congestion control is by trading the network as a black box, and thus the TCP operates by increasing its sending rate until the network becomes congested and awaiting a lost packet (timeout occurs). Therefore, the TCP introduced two variables to control network congestion of a slow start threshold (*ssthresh*) and the congestion window (*cwnd*).

The *cwnd* is the numbers of packets the sender can transmit without waiting for the receiver's *ACK*. TCP connections mainly start the transmission with a *cwnd* of one packet or one Maximum Segment Size (MSS). According to RFC 879, the default value of MSS is 576 octets (536 data stream, 20 for IP header and 20 for TCP header). Each time a new *AKC* is received, the sender increases its *cwnd* by one MSS. This phase is called the slow start phase. When the *cwnd* is equal to the *ssthresh* value, the sender increase its *cwnd* by one MSS every *RTT*. This phase is called the congestion avoidance phase. These two phases will be discussed in more detail in the following sections.

As the Internet grows in terms of number of users, data traffic, applications, and technologies, many modifications are being introduced to enhance TCP

congestion control mechanisms. These new enhancements have introduced many TCP variants during the past two decades. The following sections will review most of this effort, starting from the first TCP congestion control, TCP Tahoe, and ending in the recent TCP enhancements over the latest wireless technologies.

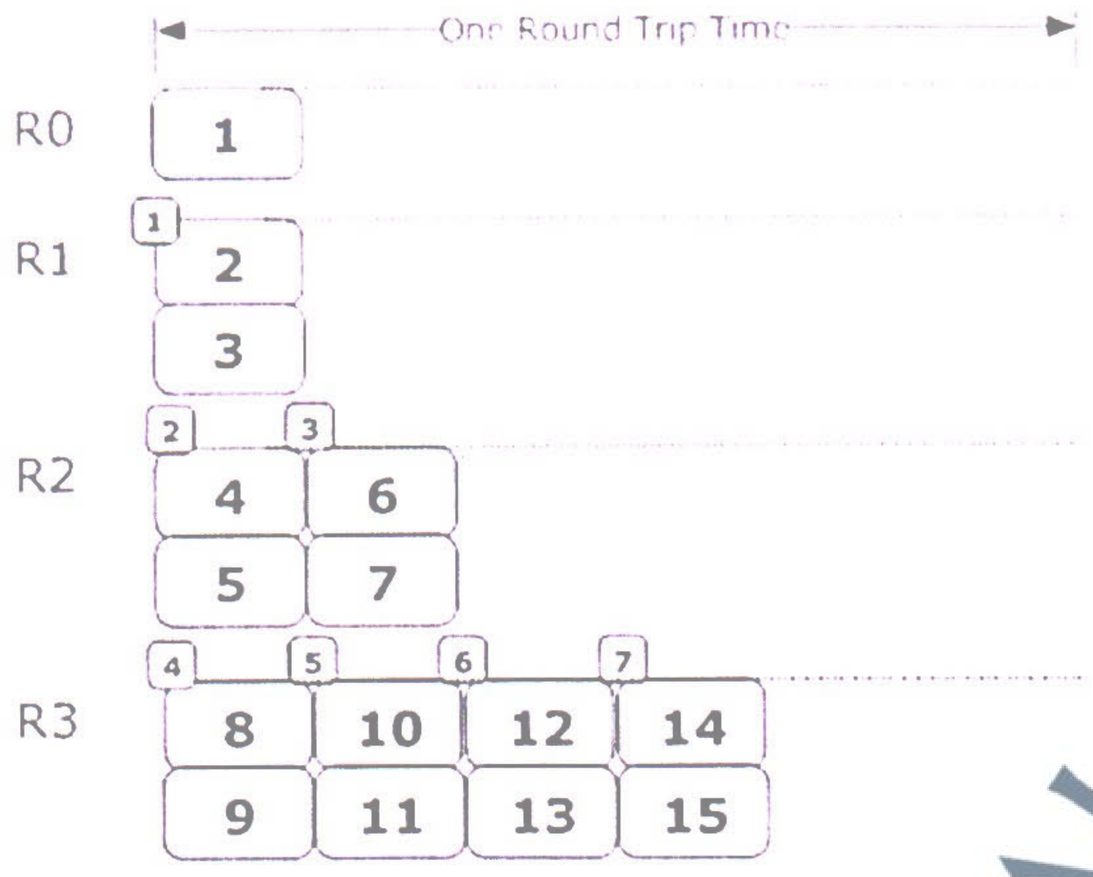
2.3 TCP Congestion Control Variants

The increased heterogeneity in network transmission media, the growth in traffic load, and unstoppable new technologies have created a dynamic research area interested in providing reliable a transport protocol to cope with these new requirements. The outcome of this huge effort has introduced a wide variety of TCP congestion control mechanisms. In this section, starting from the baseline of TCP congestion control, most of these solutions will be detailed.

2.3.1 TCP Tahoe

Tahoe was the first member of the TCP family to introduce a congestion control mechanism (Jacobson, 1988). Jacobson introduced two End-To-End (E2E) algorithms to control and avoid congestion: Slow Start, and Congestion Avoidance. TCP Tahoe calls the slow start procedure whenever a connection is started or restarted after a packet loss event (*RTO* timer expired). Slow start initiates the *cwnd* to one packet or one MSS. When each *ACK* for new data arrives at the sender side, the *cwnd* is increased by one packet, as shown in Figure 2.4.

Figure 2.4: The Chronology of Slow Start



Actually, slow start is not particularly slow, as the *cwnd* increases exponentially until the *cwnd* exceeds the *ssthresh* or congestion is observed. When congestion is observed (*RTO* timer expired), the TCP at the sender side sets the *ssthresh* value to half of the current *cwnd*, sets the *cwnd* to one packet, and calls the slow start again. On the other hand, when *cwnd* exceeds *ssthresh*, the TCP enters a congestion avoidance phase, where if *cwnd* = *ssthresh*, it either remains in slow start phase or enters the congestion avoidance phase (Bansal, 2005). In this phase, Tahoe uses Additive Increase Multiplicative Decrease (AIMD) to control the *cwnd* size.

The main philosophy behind the AIMD is to keep the *cwnd* size as high as the connection is when in equilibrium (Tahiliani et al., 2013). In AIMD, the TCP sender updates the *cwnd* in two cases: once the *ACK* is received, or when congestion is observed. For every ACK received, the sender updates the *cwnd* as:

$$cwnd \leftarrow cwnd + \frac{1}{cwnd} \tag{2.3}$$

This increment is called Additive Increase. On the other hand, once duplicate *ACKs* are received the *cwnd* is updated as:

$$cwnd \leftarrow cwnd \,/\, 2 \tag{2.4}$$

This decrement is called Multiplicative Decrease. At any time the *RTO* expires, the sender calls the Slow Start again, sets the *ssthresh* to half of the current *cwnd*, sets the *cwnd* to one MSS, and backs off the *RTO* timer. Figure 4.5 illustrates TCP behaviour in both Slow Start and AIMD phases.

Figure 2.5: Slow-Start and Congestion Avoidance Phases

One drawback of TCP Tahoe is that it requires a complete time out interval (*RTO* timer) to detect a packet loss. This issue is addressed later in TCP Reno.

2.3.2 TCP Reno

Jacobson (1990) proposed a new E2E modification to the Tahoe's congestion avoidance mechanisms: Fast Retransmission. This modification requires the receiver to send immediate *AKC* (duplicate *ACK*) when an out of order packet arrives. These duplicate *ACKs* tell the sender about a missing packet and what sequence number the receiver expects. The sender side cannot confirm a packet loss after receiving one or

two duplicate *AKCs*. However, duplicate *ACKs* can be a cause of a reordering packet arriving. According to Jacobson (1990) and Stevens (1997), the sender should wait for the third duplicate *ACK* before retransmitting what appears to be the lost packet, without waiting for its *RTO* timer to expire. Afterwards, the Fast Recovery procedure is triggered. Jacobson argued that the reception of three duplicate *ACKs* indicates more than that a packet is lost: It indicates that there is a good connection between the sender and the receiver, and the TCP wants to keep the *cwnd* as large as possible. Fast Retransmission and Fast Recovery are implemented together as follows:

1. When the sender receives the third duplicate *ACK*, the TCP sets the *ssthresh* to one-half the current *cwnd*, but not less than two MSS.
2. When the packet seems to be lost, set the *cwnd* to the *ssthresh* plus three times the MSS (3 duplicate *ACK*), and trigger the Fast Recovery procedure.
3. Each time another duplicate *ACK* arrives, increment the *cwnd* by one MSS, and transmit a packet if allowed by the new value of the *cwnd*.
4. When the next *ACK* arrives that acknowledges new data (this *AKC* should acknowledge all packets from the lost packet and the receipt of the first duplicate *ACK*), then set the *cwnd* to the *ssthresh*, and transmit the packet if allowed by the new *cwnd* new value.

Reno's Fast Recovery performs well when a single packet is dropped. However, if multiple packets from a single widow of data are dropped, TCP does not know which packet to send, or not, during Fast Recovery (Islam khan, 2012). Therefore, TCP New Reno introduced another enhancement to the Reno's Fast Recovery algorithm, and the next section covers this topic.

2.3.3 TCP NewReno

Along with other research (Hoe, 1996; Lin & Morris, 1997), NewReno (Floyd & Henderson, 1999) introduced a slight modification to the Reno's Fast Recovery algorithm. This new modification tried to solve the problem of multiple packet loss within a single window of data stream. The NewReno's Fast Recovery algorithm is implemented as follows:

1. When the 3rd duplicate *ACKs* is received and the sender is not in Fast Recovery, it records the highest sequence number transmitted in a variable called "*recover*", and sets the *ssthresh* as in the given equation:

$$ssthresh = \max(FlightSize / 2, 2 * MSS) \quad (2.5)$$

   Where *FlightSize* is the bytes already sent but not yet acknowledged.

2. The lost packet is retransmitted and the *cwnd* is set equal to the *ssthresh* plus three MSS.

3. For each duplicate *ACK cwnd* is incremented by one MSS.

4. A packet is transmitted if allowed by the new value of *cwnd*.

5. When a new *ACK* is received, if the *ACK* acknowledges all the packets up to and including the "*recover*", then the *cwnd* is set to either the min (*ssthresh*, *FlightSize* + MSS), or to the *ssthresh* value.

6. If the *ACK* is partially acknowledged (not all the packets are acknowledged up to and including the "*recover*"), then it retransmits the first unacknowledged packet, deflates the *cwnd* by the number of acknowledged packets, and adds one MSS to the *cwnd* before sending a new packet if permitted by the new value of *cwnd*.

7. The Fast Recovery procedure is not exited until an *ACK* is received that acknowledges all the packets up to and including the "*recover*".

2.3.4 TCP SACK and TCP FACK

TCP Reno and NewReno may experience poor performance when multiple packets are lost, where cumulative *ACK* provides limited information about what packets are received and/or lost. A Selective Acknowledgment (SACK) can overcome this problem (Sapra, 2015). Mathis et al. (1996) proposed a selective acknowledgment combined with a selective retransmission mechanism to retransmit lost packets only.

The SACK extension uses two TCP options. Firstly, the enabling option "SACK-permitted". This option must be sent at SYN segment to indicate that both the sender and receiver will use the SACK option. Secondly, the SACK option itself is sent once permission has been given by SACK permitted over an establish connection (Mathis et al., 1996).

On the other hand, a different algorithm known as Forward Acknowledgment (FACK) is worked on the upper option of SACK. The main idea of the FACK mechanism is that it is considered the greatest sequence number of the forward selective acknowledgment just before a packet loss.

Unfortunately, SACK and FACK have never been deployed in the Internet, as there were disagreements about how to use the SACK option in conjunction with the TCP window shift option (Arunakumari & Chennareddy, 2015).

2.3.5 TCP Vegas

TCP Vegas (Brakmo & Peterson, 1995) introduced an E2E sender side modification to the TCP Reno congestion control mechanism. In fact, TCP Vegas does not require any changes in TCP specification. Vegas achieves 30% to 70% better throughput than TCP Reno. Moreover, TCP Vegas shows friendly behaviour in comparison with other TCP connections that use different TCP implementations (Brakmo & Peterson, 1995). Rather, Vegas achieves this improvement by a more efficient use of the available bandwidth using three new techniques (Jamali, 2015).

Firstly, TCP Vegas introduced a new retransmission mechanism. Vegas reads and records the system clock each time a packet is sent and an *ACK* is received. Vegas then calculates the *RTT* based on the *ACK* arrival time and the timestamp recorded for the relevant packet. Vegas uses this value when receiving a duplicate *ACK* to decide to retransmit immediately or wait. If the value of *RTT* is greater than the value of *RTO*, then Vegas retransmits the relevant packet without waiting *n* duplicate *ACKs*. Moreover, Vegas uses this calculated *RTT* value to check when receiving the first and the second new *ACK* after a retransmission. If this value is greater than the *RTO* value, then it retransmits the relevant packet without waiting for any duplicate *ACK* (Dave et al., 2013). In other words, Vegas uses the receipt of an *ACK* as a hint to check if timeout will occur.

Secondly, Vegas introduced a new congestion avoidance mechanism as it tries to measure and then control the connection bandwidth instead of waiting for a timeout event to probe the connection bandwidth as in Reno. Therefore, Vegas calculates the expected bandwidth as follows:

$$Expexted = cwnd / BaseRTT \quad (2.6)$$

Where *BaseRTT* is the minimum recoded *RTT*. Vegas then calculates the Actual sending rate by recoding the *RTT* for a segment and then dividing the bytes transmitted by the recoded *RTT* value. Finally, Vegas uses the difference between the *Expected* and the *Actual* values to adjust its *cwnd* as follows:

- If *Expected – Actual* < α, then increase the *cwnd* linearly during the next *RTT*
- If the *Expected – Actual* > β, then decrease the *cwnd* linearly during the next *RTT*
- If α < *Expected – Actual* < β, do not change the *cwnd* value.

Where $\alpha = 1$, and $\beta = 3$ ((Brakmo & Peterson, 1995).

Thirdly, Vegas introduced a new modification to the Slow Start mechanism. Basically, Slow-Start initiates the *cwnd* by a value called the initial threshold window. However, Vegas argued that there is no safe initial threshold window size when a connection is started. Therefore, Vegas suggested using the *Expected* and the *Actual* bandwidth calculations to either use exponential (Slow-Start) or linear (Congestion Avoidance) *cwnd* incrementing.

2.3.6 TCP Westwood

TCP Westwood (Mascolo et al., 2001) is a modification of the TCP Reno congestion control mechanism to improve TCP performance over wireless networks. TCPW significantly improves the TCP performance over wired and wireless networks. TCPW introduced a new E2E bandwidth estimation mechanism using the following formula:

$$EBW = \frac{acked * MSS}{t_k - t_{k-1}} \tag{2.7}$$

Where *acked* is the number of the acknowledged packets during $t_k - t_{k-1}$period and MSS is the maximum segment size. The estimated bandwidth is used to set the *cwnd* and the *ssthresh* values after receiving three duplicate *ACK* or *RTO* expired. The main important feature of TCPW is that it does not require any feedback from the intermediate nodes to calculate the *EBW* value. Rather, it is a full End-To-End sender side modification to the standard Reno congestion control mechanism. The key idea of measuring the available bandwidth is by monitoring the returning *ACK* rate at the sender side. Friendless with TCP Reno also added more significant of adopting TCPW over wired and wireless networks. The following pseudo code shows the TCPW action once three *DupACKs* are received:

```
if (n DUPACKs are received)
ssthresh = (BWE*RTTmin)/seg_size;
if (cwin>ssthresh) /*congestion avoidance */
cwin = ssthresh;
endif
endif
/*Where BWE is the estimated bandwidth, RTTmin is
the minimum recoded Round Trip Time, seg_Size=512
bits, cwin is the congestion window.*/
```

TCPW modified the Reno's Timeout procedure to speedy recovery *ssthresh* value as shown in the following pseudo code:

```
if (coarse timeout expires)
ssthresh = (BWE*RTTmin)/seg_size;
if (ssthresh < 2)
ssthresh = 2;
endif;
cwin = 1;
endif
```

In fact, TCPW can be viewed as a new revolution in TCP congestion control mechanisms. The TCPW bandwidth estimation mechanism is particularly effective over wireless networks where sporadic packet losses occur due to radio channels being misinterpreted as a congestion symptom. A throughput improvement of up to 550% rather than NewReno is shown in Mascolo et al. (2001).

Since then, TCP congestion control mechanisms have seen extensive improvements to enhance its performance over dynamic traffic load and heterogeneous network environments.

2.4 TCP over Wireless Networks

Wireless technologies have experienced intensive improvements in the past two decades, and expansive applications and services have recently been developed. Therefore, it has become urgent to adapt the traditional TCP congestion control mechanism to new technology specifications to gain its expected performance (Liu & Lee J. Y., 2011). Many new technologies have been introduced especially in mobile communication. The latest was the Long Term Evolution (LTE), marketed as 4G. Unfortunately, LTE has not shown its expected potential due to the use of classical techniques which do not support its new requirements (Yahia, 2011; Woojin et al., 2012).

Wireless networks are characterized by a high bit rate error, sporadic connectivity, long delay, and handover. Therefore, assumptions about wired networks are not applicable to wireless networks because of the loss nature of the wireless networks environments (Maisuria & Patel, 2012).

Many TCP congestion control mechanisms have been introduced to enhance TCP performance over wireless links. Mainly, these mechanisms can be classified into three main categories: End-To-End (E2E), split connection, and link level solutions (Abed, 2013, p.59; Mahmoodi, 2009, p.45, Liu & Ren, 2015). The following sections elucidate most of these solutions in brief.

2.4.1 End-To-End Solutions

The E2E principle in congestion control mechanisms states that the congestion control procedures reside in the end hosts rather than the intermediate nodes (Tian et al., 2005). E2E solutions handle the network as a black-box; therefore, E2E solutions maintain the end-to-end semantics of the original TCP. In E2E solutions, either the sender or the receiver controls the traffic flow based on the other end's notifications (Baswaraj, 2012). Figure 2.6 illustrates this concept.

Figure 2.6: E2E solution Model

The E2E solutions try to accurately probe the available network bandwidth to achieve better performance. However, calculating the link's bandwidth is still a great challenge. Most of the proposed E2E solutions try to predict the network's bandwidth using the receiver feedback. TCPW is a good example of E2E solutions. Moreover, all the TCP congestion control algorithms discussed in section 2.3 are E2E solutions. Next, other solutions will be discussed in brief, including, SMART, FREEZE, WTCP, and TCP Real.

2.4.1.1 TCP SMART

The Simple Method to Aid Retransmission (SMART) was introduced in (Keshave and Morgan, 1997). SMART is a combination of both the N-Go-Back and the selective retransmission technique (SACK). SMART tries to build a kind of bitmask of the correctly received packets at the sender side. Using SMART, the receiver *ACKs* should have two pieces of information: the cumulative *ACK* and the sequence number of the packet's *ACK*. The sender can use this information to check which packets have been received correctly and which are missing. The sender side checks its bitmask every time an *ACK* is received, and when a gap is detected, the sender selectively retransmits the missing packets based on the missing packets' sequence number in the bitmask. However, TCP SMART perfectly handles packets losses; the sender side still assumes these losses are the result of congestion and invokes congestion avoidance mechanisms.

2.4.1.2 Freeze TCP

Freeze TCP in Goff et al. (2000) introduced an E2E solution to improve TCP performance over cellular networks. Moreover, Freeze TCP does not modify the sender side or intermediate routers; instead, the modifications are restricted at the mobile client unit or receiver side. The main idea of Freeze is to make the client manage and control both the signalling and the impeded disconnections by monitoring the signal's strength. Using Freeze TCP, the client can predict an embedded handoff or a temporary disconnection. In this case, the mobile node advertises a Zero Window Probe (ZWP) to force the sender into ZWP mode and preserve the *cwnd* size. The receiver can also send a Zero Window Acknowledgment (ZWA) if it senses an

impending disconnection to force the sender into a warning period. A reasonable warning period might be the *RTT* value.

The Freeze mechanism improves TCP over cellular networks as shown in Goff et al. (2000); however, it is required cross-layer information, and some details about roaming and handoff algorithms are implemented in the network interface card on the interface devices (Tian et al., 2005).

2.4.1.3 WTCP

Wireless Transmission Control Protocol (WTCP) is a rate-based E2E congestion control mechanism proposed in Sinha et al. (2002) to improve the TCP performance over wireless links. WTCP uses a rate-based transmission control mechanism rather than the classical window-based control used by other E2E solutions. Therefore, WTCP traffic is formally-shaped, rather than a burst traffic shape, which results in using other window-based control forms. A WTCP rate-based mechanism uses the ratio of both the inter-packet separation at the receiver and the inter packet separation at the sender side to control the TCP transmission rate. Moreover, WTCP uses Selective *ACK* to ensure it is reliability rather than using retransmission timeout.

On the other hand, WTCP uses a heuristic-based mechanism to distinguish the cause of any packet losses. For example, if packets *j* and *i* are received, and packets *i+1*, ..., *j-1*, then this heuristic is calculated as in the following equation:

$$per\_pktsep \leftarrow \frac{rece\_time_j - rece\_time_i}{j - i} \tag{2.7}$$

Where, *rece_time* is the arrival time of the last bit of the received packet. If the *per_pktsep* value is closer to the inter packet separator value at the receiver, then

WTCP assumes that the losses were random losses. Otherwise, WTCP assumes congestion losses and therefore it reduces the sending rate to half. However, the WTCP loss prediction mechanism is able to correctly predict random losses, and few congestion losses are considered as random losses using the same mechanism proposed in Sinha et al. (2002).

One consequence of using WTCP is that it is a receiver side traffic control protocol, and therefore there is a possibility of a buffer overflows before the sender is acknowledged to reduce its transmission rate.

2.4.1.4 TCP Real

This is an E2E receiver-oriented protocol proposed to enhance a TCP congestion control mechanism over heterogeneous wired and wireless networks and delay-sensitive applications (Tsaoussidis, 2002). TCP Real performs a standard congestion control mechanism instead; its receiver-oriented mechanism allows two amending mechanisms: a new congestion avoidance mechanism and an advanced error detection and classification.

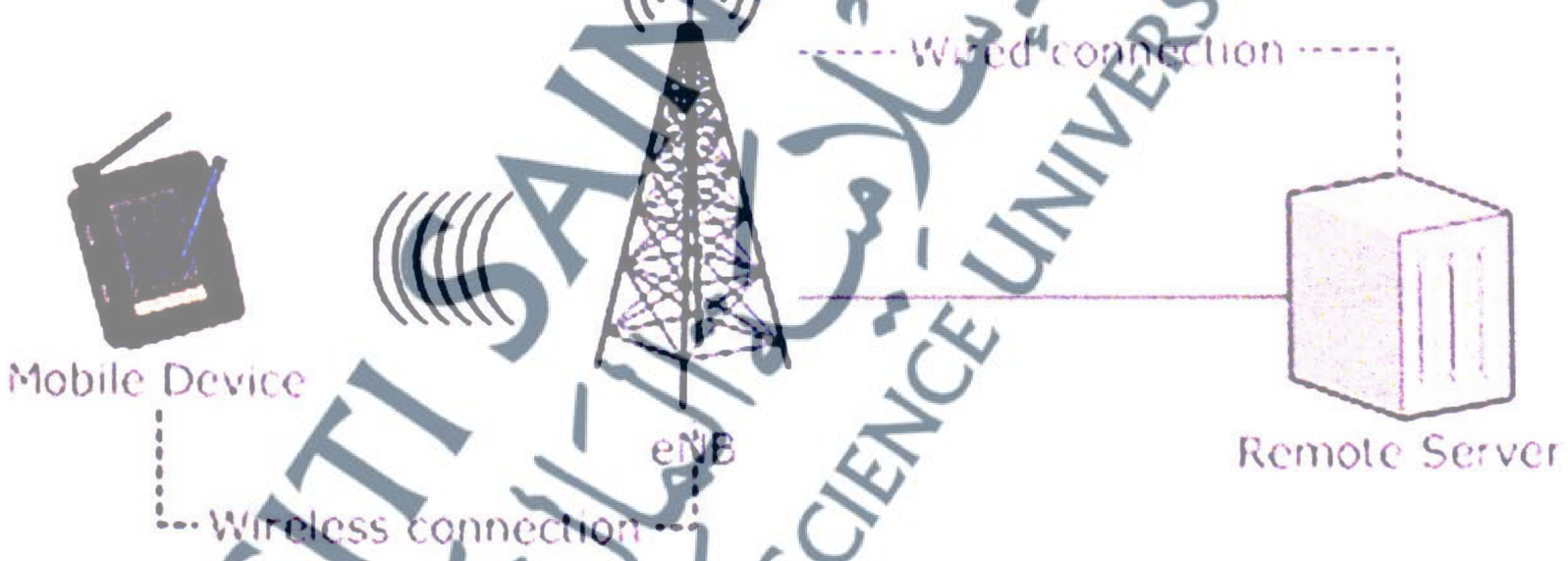
TCP Real's receiver-oriented mechanism is based on a well-known pattern of data called "waves". A TCP Real wave is effectively the window size of the traditional TCP with an additional attribute to publish its size for both peers. The receiver side computes the waves' data arrival rate. The lower the preserved rate, the smaller the congestion window suggested, and vice versa. Moreover, the preserved rate is used to distinguish transient random losses from congestion losses. This strategy is based on comparing the preserved rate with the previous rate.

In addition, TCP Real provides an improved error detection and classification strategy. Whenever the receiver observes a high rate data delivery with low jitter, packet(s) losses indicate random transient error. Therefore, congestion avoidance back-offs can be avoided. However, using a receiver traffic control may cause a buffer overflow, as WTCP does.

2.4.2 Split Connection Solutions

Based on the connection links media, split connection mechanisms have been proposed to isolate the wireless part of the network from the wired part. The main idea is to handle the TCP traffic on each side based on its characteristics, the reliable wired portion, and the lossy wireless portion, as in Figure 2.7.

Figure 2.7: TCP split connection model



Among the existing split connection solutions, the following proposals will be discussed in brief, including I-TCP, M-TCP, and SCMTP solutions.

4.2.2.1 Indirect TCP (I-TCP)

Indirect TCP was presented by Bakre and Badrinath (1995), and is a design and an implementation of I-TCP protocol which allows a mobile host to communicate with a fixed network via its Mobile Support Router (MSR). Therefore, the connection

between a fixed device (connected via a wired link) and a mobile device (connected via wireless links) should be split into two separated interactions. The first interaction is between the mobile device and its MSR. The second interaction is between the MSR and the fixed network host. Afterwards, the MSR establishes the connection on behalf of the mobile host. Whenever a handoff occurs, the new MSR takes the connection control from the previous one.

Even though I-TCP improves TCP over mobile wireless environments, its acknowledgement approach does not provide an end-to-end mechanism. Therefore, I-TCP does not provide any feedback to the sender side about the actual received time of the packet. Moreover, I-TCP requires some modifications of both the mobile host and the MSR transport layers.

2.4.2.2 Mobile TCP (M-TCP)

Brown and Singh (1997) introduced another split connection solution to enhance TCP over mobile networks. The proposed protocol was designed to preserve the following principles:

- To improve transport protocol performance over mobile networks
- To preserve the TCP end-to-end semantic
- To handle the consequences of frequent disconnections
- To adapt to dynamic bandwidth changes
- To ensure an efficient handoff procedure

Thus, M-TCP uses a bandwidth management module to determine the bandwidth size that should be located for each connection within each cell. This amount of bandwidth is periodically changed based on another mobile client's requirements and

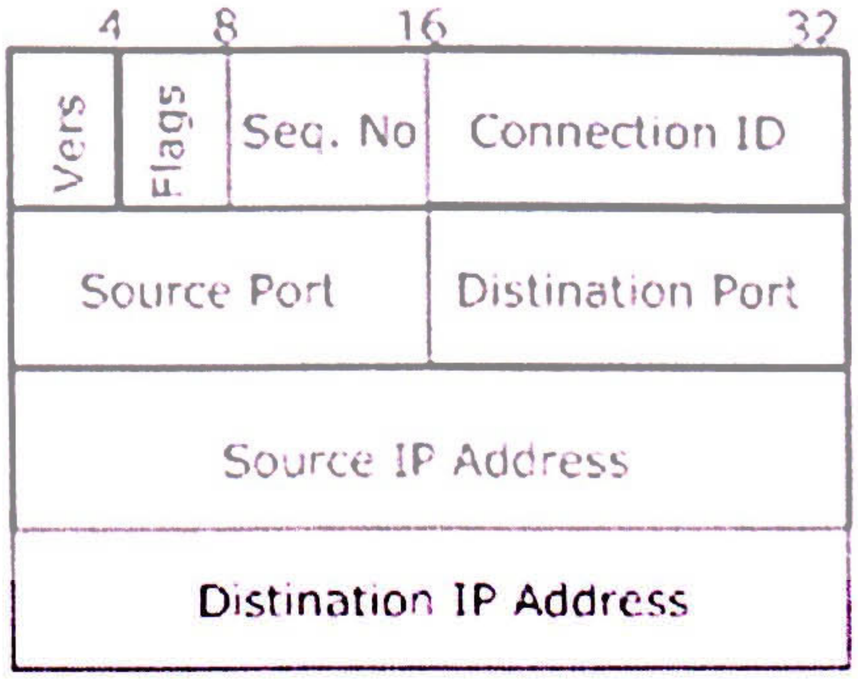
the current status of the mobile connection. Since the bandwidth amount is already fixed for each mobile client, the M-TCP does not implement a sophisticated congestion control mechanism over the wireless portion of the connection.

Unlike I-TCP, M-TCP does not acknowledge any packets until the destination does to preserve the end-to-end semantics of TCP. Moreover, M-TCP requires less handoff information to be exchanged between the mobile stations than the I-TCP (Mahmoodi, 2009, p.47). On the other hand, M-TCP will only work over a low bit error rate (Tian et al., 2005).

2.4.2.3 Mobile-End Transport Protocol (METP)

Kuang and Tripathi (1998) proposed replacing the transport layer of the wireless hosts with a new light header Mobile End Transport Protocol (METP) which is designed to run over the link-layer. The proposed METP shifts all the communication functions with a remote host from the mobile hosts to the base stations. METP takes advantage of this, and the link between the mobile host and its base station will be the first or the last hop along the data path. Therefore, METP uses a smaller header than the original TCP/IP uses, since the mobile host does not perform any datagram functions. However, the mobile host still performs a simple multiplex/demultiplex mechanisms. Figure 2.8 shows a full METP header format.

Figure 2.8: METP Protocol Header



Furthermore, METP exploits the link layer acknowledgments to quickly recover losses over wireless links. The proposed mechanism adds a new medium *ACK* method to allow the MAC level to immediately recover lost packets. This mechanism also provides ordinary transmission which guarantees a reliable delivery over the wireless portion (Kuang & Tripathi, 1998).

2.4.3 Link Level Solutions

Link level solutions suggest utilizing the link layer's reliability to shield the transport layer from the lossy nature of the wireless links. These solutions were proposed to enhance TCP performance over wireless links without modifying the transport layer protocols (Boukerche, 2006, p. 29-690). The link layer protocols operate independently from other higher layer protocols. There are several link level solutions techniques in the literature, and we describe some of those solutions in the following section.

2.4.3.1 Transport Unaware Link Improvement Protocol

The Transport Unaware Link Improvement Protocol (TULIP) was designed to enhance TCP over wireless links without modifying the transport layer protocol (Parsa & Garcia-Luna-Aceves, 2004). TULIP was designed to fit half-duplex radio links and to provide a MAC acceleration feature to improve the wireless networks throughput without changing TCP at the peer connection and without requiring proxies between them. TULIP provides a reliable service for TCP packets, and an unreliable service for other traffic types such as UDP packets. One main feature of the proposed mechanism is its ability to provide a local recovery service for all the lost packets to prevent the TCP costly retransmission mechanism.

The reliability of the TULIP is guaranteed by retransmitting missing packets regardless of what happens on the transport layer (Assaad & Zeghlache, 2007, p.145). The link layer delivers the received packets in sequence to the transport layer to avoid duplicate acknowledgements and unnecessary congestion control procedures. On the other hand, the sender detects lost packets using a bit vector returned by the receiver *ACKs*.

This proposed TULIP mechanism presents several disadvantages. First, the use of such a scheme does not solve the problem of the incompatible setting of the transport layer and link layer timers. Moreover, when receiving an erroneous packet, delaying the subsequent correctly received packets at the link layer results in more timeouts at transport layers (Assaad & Zeghlache, 2007, p.145).

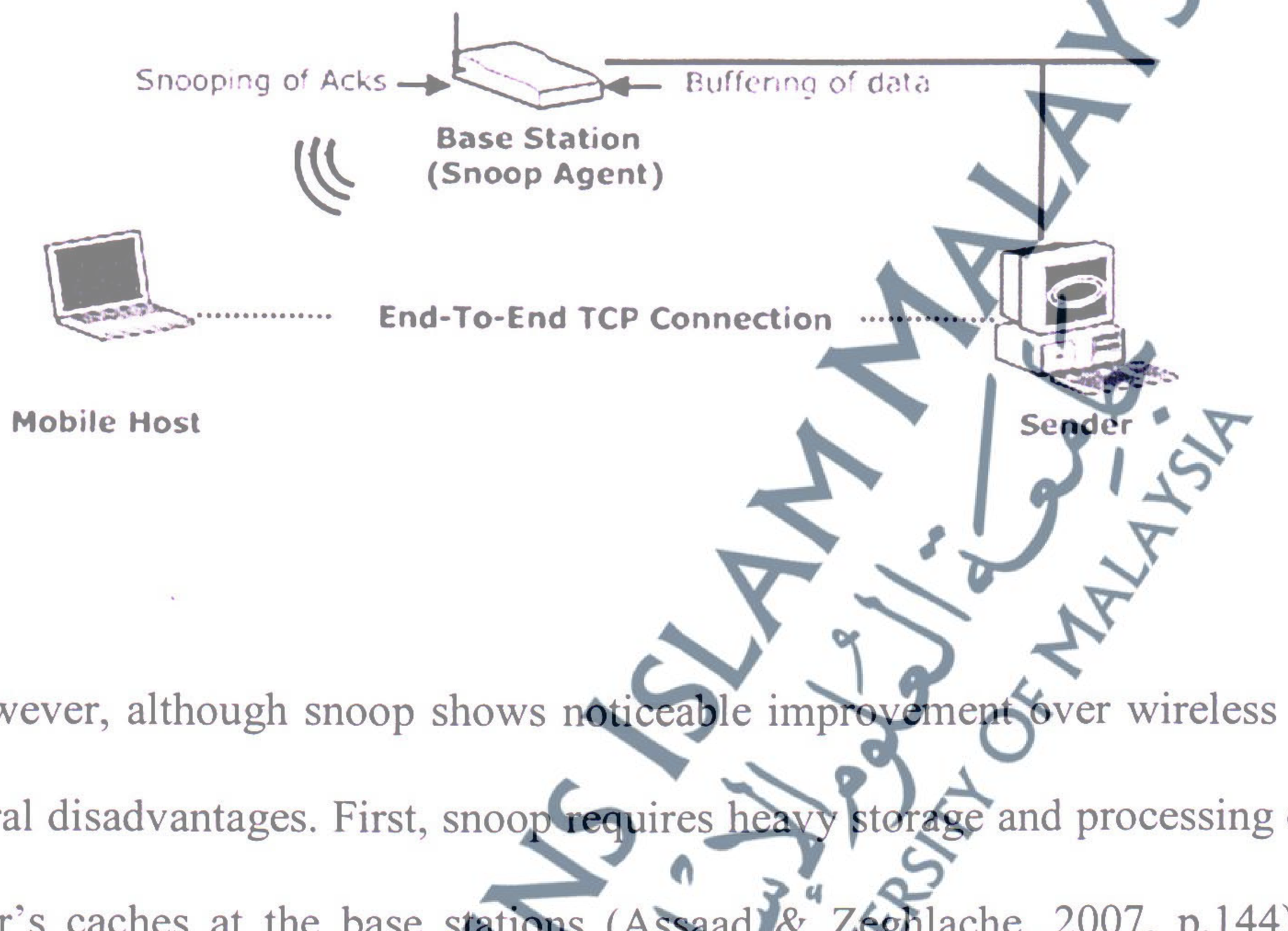
2.4.3.2 Snoop

The Snoop protocol (Amir et al., 1995) was proposed to improve TCP performance over wireless links without changing TCP implementation at the end hosts. Instead, Snoop suggested modifying the network layer software at the base stations only. The modifications were made to the router code to catch some data meant for the mobile host, and proposed new policies for the retransmission and acknowledgements mechanisms.

The base station routing code modification includes adding a new layer called the snoop layer. This layer monitors every packet passing through the base station in either direction. It then maintains a cache of all the sent packets but not yet acknowledged by its destination. When a packet loss is detected at the base station, the snoop retransmits the lost packet unaware of the packet sender. Thus, Snoop prevents

the unnecessary transport layer congestion control mechanism at the sender side (Suryanto et al., 2015). Figure 2.9 shows a typical Snoop approach.

Figure 2.9: Snoop approach



However, although snoop shows noticeable improvement over wireless links, it has several disadvantages. First, snoop requires heavy storage and processing of the snoop layer's caches at the base stations (Assaad & Zeghlache, 2007, p.144), and second, snooping and buffering caches may be useless if the end hosts apply a certain encryption scheme (Boukerche, 2006, p.29-691).

2.4.3.3 Forward Error Correction Solutions

Forward Error Correction (FEC) is a well-known technology that allows error correction at the physical layer ("Silver Peak Systems Inc.," n.d.). FEC is based on adding redundancy to transmitted messages so that the receiver can detect and recover common transmission failures. On the other hand, error detection techniques do not provide any way to locate or correct errors instead; it requires the retransmission of the block of data if errors are detected within that block (Chockalingam et al., 1999).

FEC uses complex functions to encode redundant original bits. If the original bits appear literately in the encoded function then it is systematically encoded; otherwise, it is non-systematic encoding. Basically, the two main types of FEC codes are block code and convolutional code. The block codes work on a fixed-size blocks of bits, while the convolutional code works on arbitrary length of bits or symbols.

Many studies have been conducted to use FEC techniques to enhance the transport layer over high bit error networks. In Araar et al. (2005), they introduced a reliable multicast protocol for dynamic wireless network environments based on an improved FCE coding technique. The improved FEC used in the dynamic servers to allow its members to leave and join the group anytime.

Tang & McKinley (2002) introduced a protocol called the Adaptive FEC-based Reliable Multicast (AFRM). The proposed protocol is based on NACK and uses both the reactive FEC and the proactive FEC as relevant parameters to adapt the protocol behaviour to the network conditions. In addition, the AFRM flow control uses a novel method to recognize the queuing packets losses from other losses.

Even though FEC-based solutions show good performance over wireless links, these solutions are rarely used. The encoding/decoding overhead is the main reason to avoid using such solutions.

2.5 SCTP

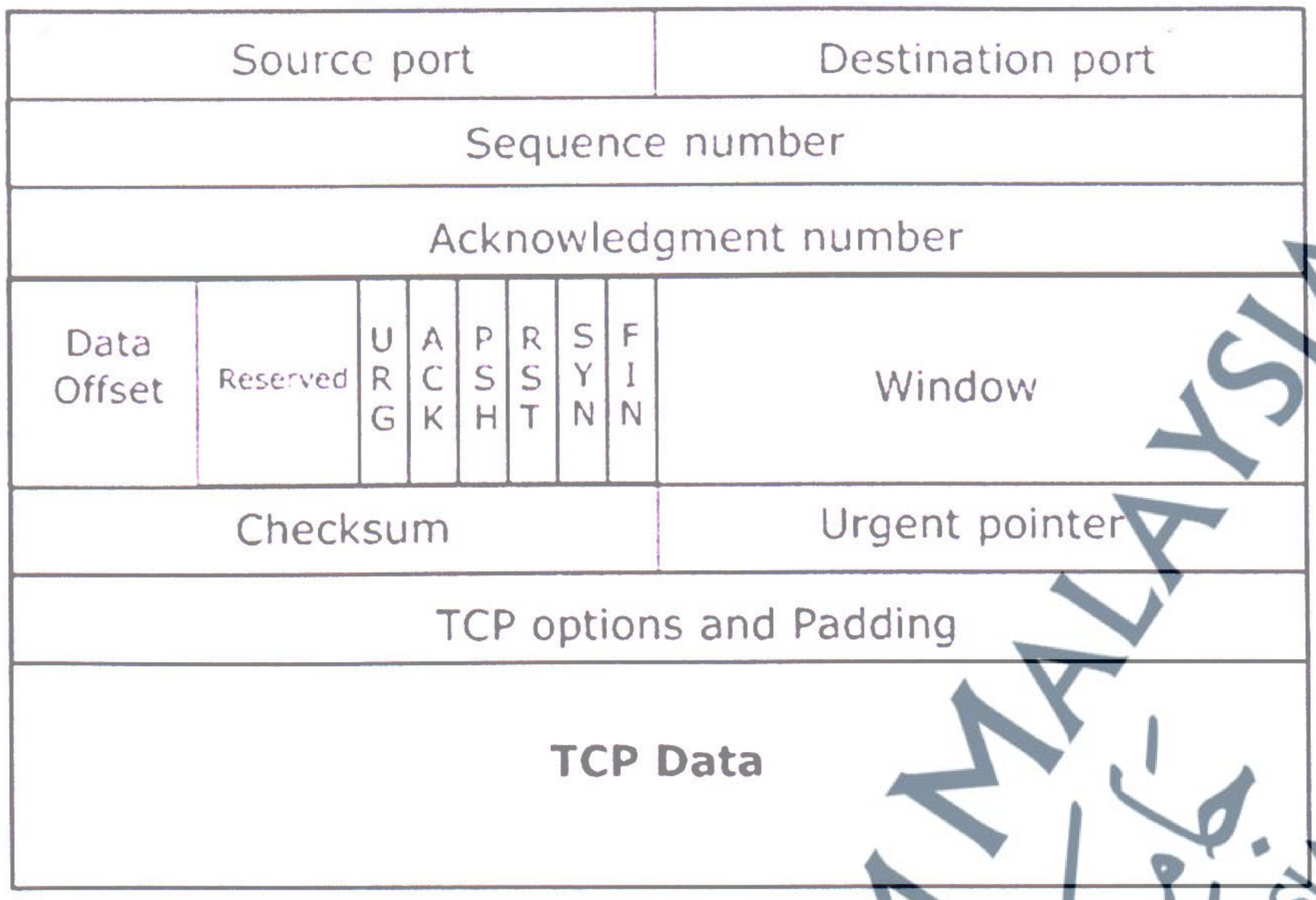
As mentioned in section 2.1.1 of this chapter, Stewart et al. introduced the Stream Control Transmission Protocol in RFC 2960. The new protocol was designed to provide reliable transmission service on top of the unreliable IP for Public Switched Telephone Network (PSTN). Then, in 2000, the Internet Engineering Task Force

(IETF) embraced SCTP in the transport layer protocols in TCP/IP, which were stacked together with TCP and UDP (Stewart & Metz, 2001).

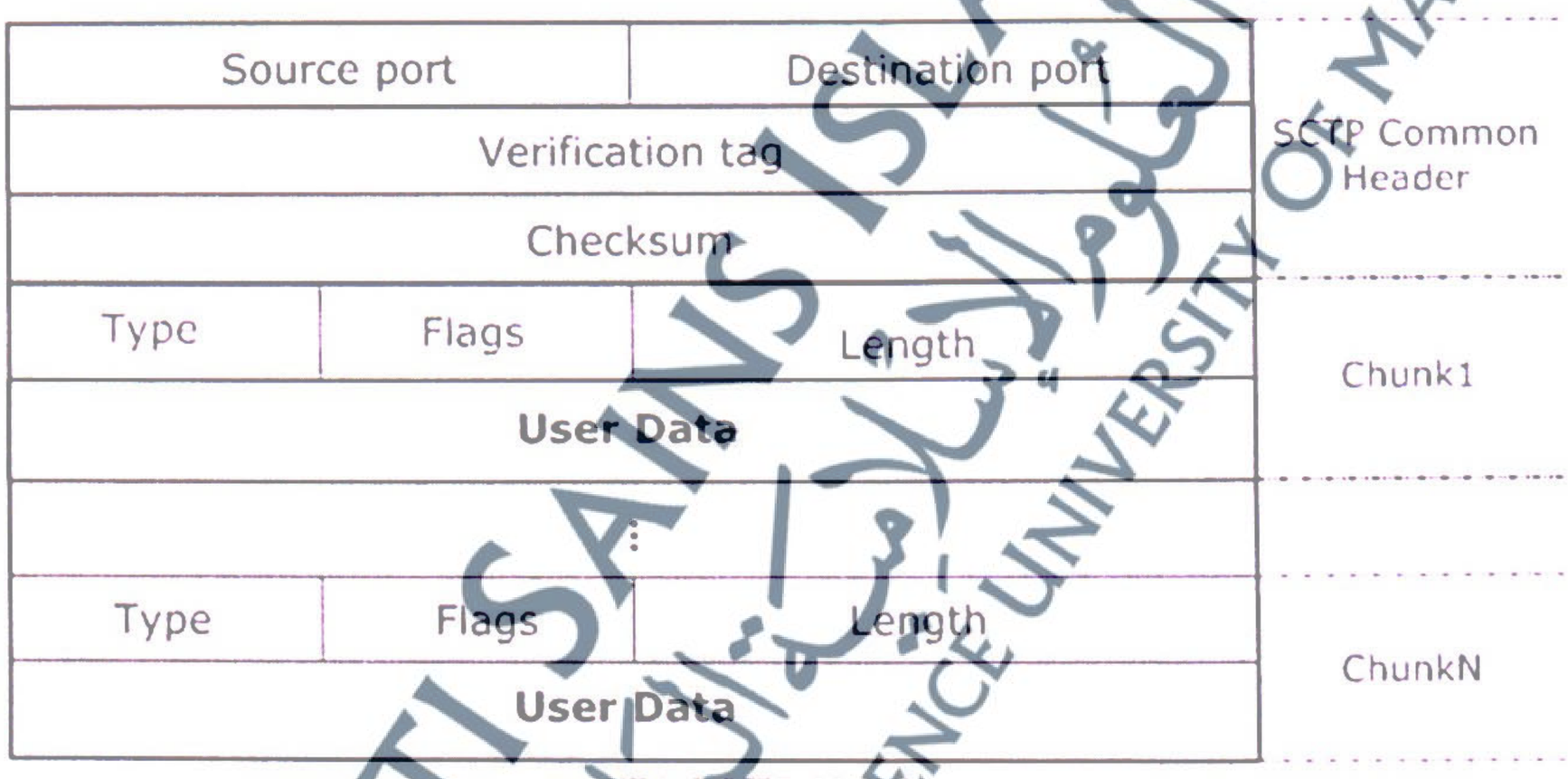
SCTP offers basic TCP services, including reliable delivery, point-to-point, and connection-oriented transport services over IP networks. Also, it shares most TCP functions, such as flow control, congestion control, data recovery, and error handling functions. Moreover, SCTP preserves the TCP message boundary by bundling the messages into one or more SCTP chunks. In addition, the SCTP protocol header shares the common TCP header, as Figure 2.10 shows. Therefore, any application running over TCP can be ported to run under SCTP without loss of function (Stewart & Metz, 2001).

On the other hand, there are several differences between the two protocols. The most decisive differences are the multi-homing and the partial-ordering mechanisms provided by SCTP. Multi-homing enables SCTP hosts to establish a session with other SCTP hosts over multiple interfaces defined by different IP addresses. In other words, an end-point is considered as multi-homing if it has multiple destination addresses to reach that end-point (Stewart, 2007). This mechanism allows SCTP hosts to send outstanding data if the primary path becomes inactive (e.g. there is a failure) using an alternate active destination address if one exists.

Figure 2.10: TCP and SCTP Headers



(A) TCP Header



(B) SCTP Header

TCP delivers data in a strict order, whereas some applications require partial-ordering data delivery. Moreover, other applications require reliable data transmission with no concern about sequence delivery. In either case, TCP's order-delivery head-of-line blocking causes unnecessary delay. On the other hand, SCTP's partial-ordering delivery provides in-order delivery of one or more related sequences of packets. Thus, SCTP is useful for applications that need both reliable delivery and fast processing of unrelated data streams.

To continue with the differences between the two protocols, SCTP uses the term of association to define the peer hosts transmission state. As mentioned above, end-point associations can employ one or more addresses at each end host where SCTP supports multi-homing. Figure 2.11 shows SCTP association architecture.

Figure 2.11: An SCTP association [adopted from Stewart et al. (2000)]

SCTP uses a four way handshake mechanism to establish an SCTP association between the two hosts. In contrast to the TCP's three way handshake, SCTP's four way handshake eliminates the problem of TCP Dos attacks (Stewart & Metz, 2001). Thus, during the association initialization, SCTP employs a security cookie to mitigate the impact of TCP SYN flooding attacks. On the other hand, SCTP uses the TCP's three way handshake to shut down an association, but with one difference: SCTP does not support a half-closed state, which allows the connection peer to exchange data after carefully shutting down.

2.6 Conclusion

This chapter has presented a background study on common transport protocols in communication systems. The first section introduced a brief introduction on network layering concepts. Secondly, TCP fundamentals were discussed in section two. Thirdly, we presented a detailed study of the common end-to-end TCP implementations. Table 2.1 summarized the main specifications of Tahoe, Reno, NewReno, and TCPW implantations. Fourthly, section four presented the literature on TCP congestion control mechanisms over wireless networks. Finally, in section five, we introduced the SCTP protocol.

Table 2.1: Standard TCP variants comparison

	Tahoe	Reno	NewReno	TCPW
Slow Start phase	√	√	√	√
Congestion Avoidance phase	√	√	√	√
Fast Retransmission	×	√	√	√
Fast Recovery	×	√	√	√
Bandwidth Estimation method	×	×	×	√
Method to recover multi packet losses	×	×	√	√
Method to recognize packet losses causes	×	×	×	×
Method to set up the Initial ssthresh value	×	×	×	×
Incrementing the sending rate based on bandwidth utilization	×	×	×	×

√- support, × not support

The next chapter will present the methodology used for carrying out this study. The research tools and the evaluation methods of the proposed congestion control mechanisms will be discussed in detail.