EUROPEAN ACADEMIC RESEARCH
Vol. IV, Issue 8/ November 2016

Impact Factor: 3.4546 (UIF)
DRJI Value: 5.9 (B+)

ISSN 2286-4822
www.euacademic.org

Security Evaluation of the Adaptive Congestion Control Algorithms for Virtual Data Center Communication

VINCENT O. NYANGARESI
Dr. S. OGARA
Dr. S. ABEKA
School of Informatics and Innovative Systems
Jaramogi Oginga Odinga University of Science & Technology

Abstract:

Virtualization has been vastly implemented in many organizations, facilitating the installation of more than one operating system in one physical machine. This helps save on costs associated with the physical hardware. This technology is employed in data centers which hold mission critical resources for organizations. This requires that there be secured and uninterrupted communication between various terminals and these centers. Congestion control algorithms have emerged as ideal mechanisms of preventing the sender from overwhelming the receiver and other network entities with data packets. However, these algorithms in their current operations in detecting network congestions and responding to it, can be utilized for malicious activities. As an illustration, the TCP acknowledgements that are employed by some congestion control algorithms to detect network congestion can be employed to launch SYN flooding attacks. Continuous SYN flooding attacks lead to denial of services. Both SYN flooding and DOS raise security challenges in networked enterprises. Therefore, poor detection and handling of network congestion can compromise the security of the data in transit. This paper provides an in-depth evaluation of the various congestion control algorithm such as slow start, congestion avoidance, fast retransmit and fast recovery. Since these protocols are normally implemented in TCP variants, a

review of TCP variants such as TCP Tahoe, TCP Reno, TCP Vegas, TCP CUBIC, BIC TCP, TCP Hybla, TCP Westwood and Compound TCP is provided. The ultimate goal was to demonstrate that even though they are implemented in a wide variety of networks, these algorithms are deficient in a number of ways. For instance, in TCP Tahoe, when congestion is detected, the congestion window is set to one maximum segment size. This catastrophic reduction in data rates can greatly affect the ongoing communications, such as video transfers. Moreover, all of the TCP variants employ acknowledgements either to establish a connection, detect packet loss or to react to packet losses. Attacks such as TCP SYN have been shown to take advantage of these acknowledgements during the three way handshake to launch denial of service attacks. The significance of this study then lies on the fact that based on the observed discrepancies; it justifies the development of a novel congestion control algorithm to address the identified setbacks. The results clearly indicated the inefficient utilization of the network bandwidth by the existing congestion control algorithms and the ease with which intruders can deny availability to the legitimate users by flooding the senders with acknowledgements during the three way handshake phase.

Key words: Congestion control, algorithm, TCP variants, fast retransmit, fast recovery, slow start

I. INTRODUCTION

The TCP protocol stack utilizes four algorithms to manage congestion in computer networks. These four algorithms are: slow start, congestion avoidance, fast retransmit and fast recovery. Every TCP communication involves the initial three way handshake, during which the sender transmits SYN segment to the receiver. This synchronization packet has its SYN flag bit set.

Upon the receipt of the SYN packet, the destination machine responds with a SYN/ACK, which is a packet whose SYN and ACK flag bits are set. The sender then returns an

ACK segment for the received SYN/ACK data segment received from the destination machine, after which the user payload can be sent over the communication channel. An attack exploiting the three way handshake is the TCP SYN flooding.

In this attack, the intruder avoids sending back the required acknowledgement data segment. When the connection for a given port gets times out, another TCP SYN request is sent to source address, using the same port number. The source is now engaged in communication where the receiver (intruder) continuously transmits a stream of SYN segments and never wants to correctly acknowledge the SYN/ACK data segment.

In so doing, the attacker establishes endless half-open connections with the source. In case of a virtual data center environment that holds critical organizational resources such as databases and server farms, the effect of this is that legitimate users are denied access to these resources.

A. SLOW START ALGORITHM

This algorithm is utilized establish packet transmission and to detect the size of the available bandwidth in order to avoid the transmission of large bursts of data that may fill up the network. This phase uses two extra variables, namely the congestion window and the start threshold values.

Whereas the congestion window sets the maximum amount of packets that the sender may send out into the communication channel before receiving acknowledgement, the slow start threshold value determines when to enter or exit the slow start phase and enter the congestion phase. In their paper, Kire et al., (2016) pointed out that the maximum amount of packets that a sender may transmit is governed by the relation below:

$$min\{cwnd, rwnd\} \quad \text{(i)}$$

where *cwnd* is the transmitter congestion window and *rwnd* is the receiver's advertised window.

According to Christos et al., (2012), during the slow start phase, the initial value of the congestion window (*cwnd*) is set to one segment, where one segment is equal to maximum segment size (MSS) value:

$$cwnd = 1\ segment \quad \text{.....(ii)}$$

Where 1 segment = *MSS* bytes.

Each time the transmitter receives an *ACK*, it increases its congestion window by *MSS* bytes.

$$cwnd = cwnd + MSS \quad \text{.....(iii)}$$

Therefore, as long as the *cwnd*<*sstresh*, the TCP communication is still under the control of the slow start phase. The *cwnd* is essentially based on the transmitter's own assessment of the communication network while *rwnd* is determined by the size of the receiver's incoming buffer size.
Evidently, the slow start phase experiences an exponential increment as demonstrated by Figure 1. The Y- axis represents the transmitter congestion window (*cwnd*) while the X- axis represents the elapsed connection time.

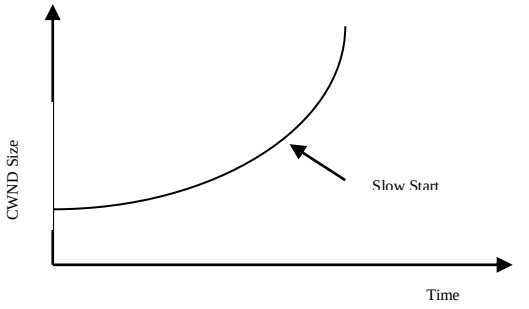

Figure 1: The Slow Start Algorithm

On the same breadth, when two segments are transmitted and each of them is acknowledged, the congestion window is increased to four. However, as Jonathan (2012) illustrates, the slow start may not be entirely exponential in actual sense since in some situations, the receiver may delay its acknowledgements, and therefore end up sending one acknowledgement for every two segments it receives.

B. CONGESTION AVOIDANCE ALGORITHM

In a TCP environment, when three duplicate acknowledgments are received, the sender translates this to mean that packet loss has occurred. Therefore, it abandons the slow start phase and enters the congestion avoidance phase (Simon & Dimitrios, 2016). This phase then persists on every incoming non-duplicate acknowledgement. The sender's congestion window increment is governed by relation:

$$cwnd = cwnd + mss/cwnd \quad \text{.....(iv)}$$

This channel congestion avoidance phase is entered when *cwnd>=sstresh*. Evidently, this is a linear congestion window growth as illustrated by Figure 2.

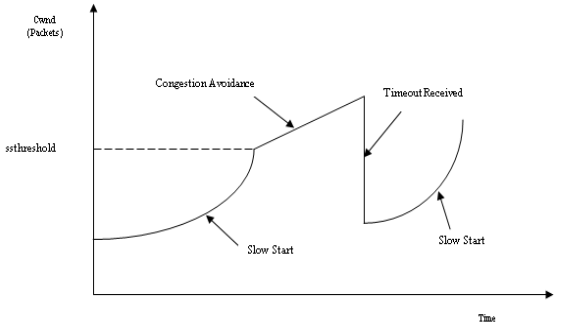

Figure 2: Congestion Avoidance Algorithm

C. FAST RETRANSMIT ALGORITHM

In circumstances where a data segment arrives at the receiver out of sequence, a duplicate acknowledgment (*DUPACK*) is immediately sent to the transmitter. The *DUPACK* serves, to

inform the corresponding receiver of this abnormally and in addition, instruct the sender the sequence number of the data packet that the receiver is expecting. In normal situations, *DUPACKs* can be generated due to dropped segments, re-ordering of data segments and the replication of acknowledgements or data segments (Nishida, 2012). In the case of dropped segments, *DUPACKs* will force the transmitter to send all segments after the dropped one.

Packet reordering is the scenario that arises when network packets from the same transmitter arrive at the destination in a different order from the order in which they were sent from the source. According to Callegari et al., (2012) this re-ordering may be occasioned by link layer recovery, handovers, priority blocking, route changes, multipath routing, and differential services.

Unfortunately, due to its structural weaknesses, TCP does not discriminate against *DUPACKs* caused by lost data segments and the reordering of segments (Dong & Schapira, 2015). As such, it waits for a limited number of *DUPACKs* to be received and works on the assumption that a reordering of segments will be only one or two *DUPACKs* before the reordered segment is processed. It therefore, anticipates that a new *ACK* will be generated after the re-ordered segment if processed. However, upon the reception of three or more *DUPACKs* in a row, it interprets this to be a strong indication that a segment has been lost, which might not actually be the case. It thus performs a retransmission of the apparent missing segment, without waiting for the retransmission timer to expire. This constitutes the fast retransmit phase.

D. FAST RECOVERY ALGORITHM

Once the TCP fast retransmit resends what it considers to be the missing segment, congestion avoidance is entered instead of the slow start is performed (Stefan et al., 2015). This constitute the fast recovery algorithm which then takes control of the

network transmissions of the preceding new data packets until a non-duplicate acknowledgement arrives at the transmitter. Figure 3 gives a pictorial representation of fast retransmit and fast recovery algorithms.

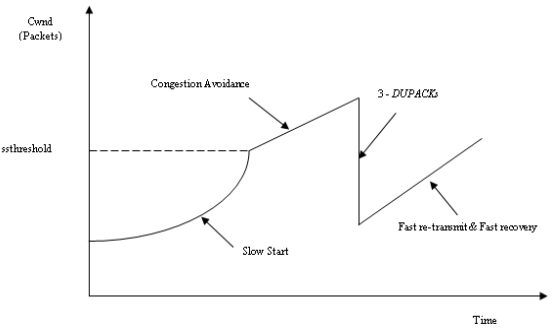

Figure 3: Fast Retransmit and Fast Recovery Algorithms

Comparing Figures 2 and Figure 3, then it becomes apparent that after the sporadic fall in transmission rate represented by *'Timeout Received'* in Figure 2, the graph assumes an exponential growth confirming that the slow start phase has been entered. However, for the case of Figure 3, after the sporadic fall which is represented by *'3- DUPACKs'*, the graph assumes a linear growth, implying that the congestion avoidance algorithm has taken over control of the transmissions.

A close observation of the above graphs reveals that slow start algorithm may lead to underutilization of the available bandwidth, especially during its initial set up phase in which small number of packets are sent (Shaneel, 2014). In essence, it takes long before it starts to fully utilize large network bandwidths, such as those provided by fiber optic channels. The rest of the bandwidth can therefore be employed by attackers to infiltrate large volume information from victim computers. Moreover, according to Guang (2015), when congestion is detected, the congestion window is adjusted to very small values which may adversely affect the ongoing communications, such as those involving video streaming.

In their analysis of end to end adaptive algorithms, Christos et al.,(2012) noted that TCP takes packet loss as an indication of congestion in computer networks. This is erroneous since packet loss may occur due to factors such as fading, shadowing, hand off and other radio effects. Therefore, these factors although not linked to congestion, cause the congestion window to be adjusted downwards inappropriately. Guang (2015) explains that through SYN flooding attacks, the sender's congestion window can also be increased for malicious reasons.

II. RELATED WORK

Poor detection and handling of network congestion can compromise the security of the data in transit between the various terminals and the virtualized data centers. The data centers hold some of the most valuable organizational resources and therefore adequate measures should be taken to protect them. This means that there be secured communication to and from these server farms. Since most data transfers utilize the TCP connections, it is imperative to investigate how the various TCP variants detect and deal with congestion.

All the adaptive congestion control algorithms, which include slow start, congestion avoidance, fast retransmit and fast recovery are implemented differently in the TCP variants (Shohidul et al., 2009). These TCP variants include TCP variants, such as TCP Tahoe, TCP Reno, BIC TCP, TCP CUBIC, TCP Vegas, TCP hybla, TCP Westwood and Compound TCP. For example, TCP Tahoe implement slow start, congestion avoidance and fast retransmit. On the other hand, TCP Reno implements all of the four algorithms. Mudassar et al., (2016) provided an extensive analysis of the congestion algorithms by investigating the TCP variants.

The security challenge of TCP connections arises from the fact that when congestion crops up, then some packets are dropped and the dropped packets must be retransmitted. This

means that an intruder may launch a denial of service attack which effectively overwhelms the receiver and hence the receiver ends up dropping some packets. The consequence of this is that an intruder may force the transmitter to retransmit packets sent earlier on, which can then be redirected and used for malicious activities(man-in –the –middle attacks employ this weakness).

An example of this attack is the SYN Flooding Attack (Robbie, 2015). The SYN flood is a form of denial of services (DOS) attack in which an intruder sends numerous SYN requests to a victim's TCP port. However, a trespasser has no intent of finishing the 3-way handshake process.

Bellovin (2013) states employing this attack, an impostor can continuously flood the victim's buffer, by new connections request but never fully establishing the connection. This attack means that the bandwidth utilization mechanisms and causes of congestions in computer networks should be evaluated, failure of which the congestions can be attacker-initiated for malicious activities.

One of the ways of investigation the causes of congestion in networks are to examine how the available bandwidth is being utilized. John and Barry (2013) illustrates that the originally TCP was designed for early, low bandwidth, short distance networks. This means that the standard TCP cannot utilize the maximum bandwidth provided in current high bandwidth network environments.

To address this gap, TCP variants have been established for current elongated distance high bandwidth networks.

III. RESEARCH APPROACH

In this paper, the virtual data center was simulated by using the Oracle VirtualBox VM as the hypervisor, Windows Server 2012 as the guest operating system and Windows 2007 Professional as the host operating system. The TBIT (TCP

Behavior Inference Tool) software was installed in one of the terminal machine that was utilized to access the virtualized server. The role of this software was to produce the required congestion window – time graphs that were the subject of evaluation in this study. The simulation and experimental setups were conceptually organized as shown in Figure 4.

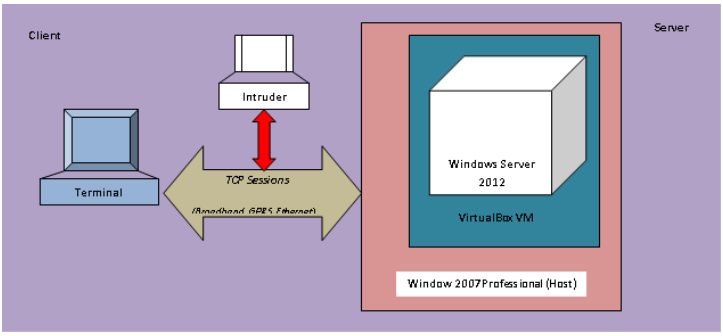

Figure 4: Conceptual Experimental Setup

The idea was to transfer some data files from the Windows Server 2012 located in VirtualBox Hypervisor, into the terminal machine. To accomplish this, TCP sessions were employed, using port 80, which is the standard HTTP port for data transfer. In this arrangement, the combination of VirtualBox, Windows Server 2012 and the host operating system represented the virtual data center. These TCP sessions are not immune from intruders who might be listening, or inject packets into the channel.

Figure 5 that follows shows the physical arrangement of the experimental and simulation setup. Linux operating system was installed in the client and this environment facilitated the installation and usage of a set of rich network monitoring and analysis tools. During the course of data transfer, TBIT software was utilized to monitor the TCP congestion window (*cwnd*) of the sender and the receiver advertised window

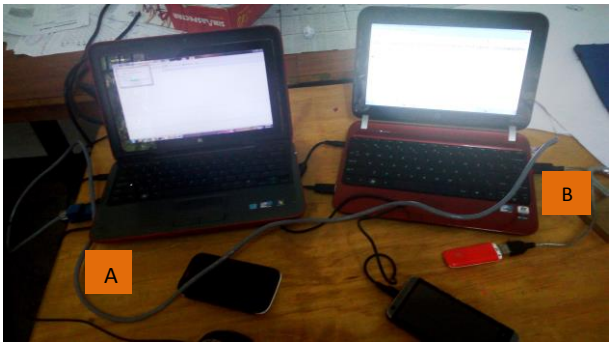

Figure 5: Physical Experimental Setup

(*rwnd*). The TCP segments were captured at the terminal using *tcpdump* tool. Thereafter, the data was analyzed using *tcptrace* and *xplot* programs. While machine A was the client, machine B was the virtualized server.

As this figure shows, the two machines were connected by an Ethernet cable. During the course of the investigations, a hotspot was established at the server, which the client connected to via the broadband modem (*shown in red*). Moreover, the mobile GPRS was also employed to provide a Tethering and portable hotspot, and the two machines were configured to use this mobile hotspot for data exchange. Therefore, the TCP variants were investigated over the Ethernet connections, broadband connections and mobile GPRS tethered hotspot.

IV. RESULTS

It was noted that based on the type of TCP implementation, the behaviors of the congestion – time graphs were different. This could be attributed to the fact that the TCP variants implement or fail to implement the following congestion control algorithms: *Slow-Start, Congestion Avoidance, Fast Recovery and Fast Retransmit*.

A. TCP Tahoe

In their paper, Mudassar et al., (2016) explained that in TCP Tahoe, congestion is detected by the receipt of *DUPACKs* or when the sender experiences a timeout. It responds to congestion by adjusting the congestion window to 1 MSS (whose size is 1500 bytes) and set the slow start threshold to half the size of the current congestion window. Figure 6 that follows shows a plot of congestion window against time in a TCP connection.

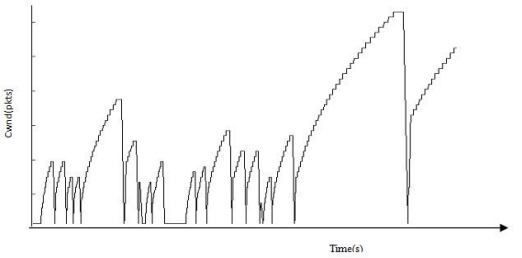

Figure 6: TCP Tahoe *Cwnd* Adjustments against Time

Figure 6 shows that the size of the congestion window increases exponentially until packet loss is encountered. At this instant, the size of the congestion window is set to one MSS value. The size of the slow start threshold value is set to half the current congestion window size. Afterwards, the congestion window is incremented until another packet loss is encountered.

B. TCP Reno

The TCP Reno implements all the congestion algorithms in TCP Tahoe, but also includes the fast recovery algorithm. Jonathan (2012) discusses that in this variant, network congestion is identified through a packet loss using the Re-transmit Time Out (RTO) and not by three *DUPACKs*. When congestion occurs, the value of congestion window (*cwnd*) is adjusted to half of its prior value. Figure 7 shows the graph of congestion window against time. In circumstances where the source is still

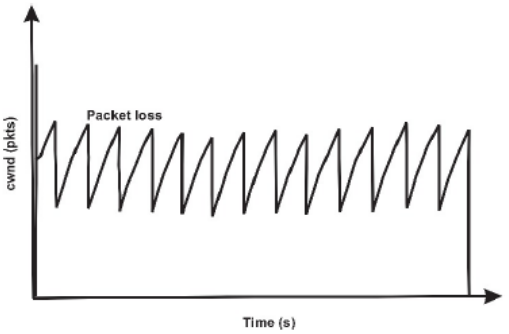

Figure 7: TCP Reno *Cwnd* Adjustments against Time

capable of receiving the acknowledgements and after receiving a number of *DUPACKs*, this variant enters into the fast recovery phase. This forces transmitter to re-send the lost packets. Another distinguishing feature of this approach is that will never fall back into slow start state, something that TCP Tahoe does.

C. High-Speed TCP

In this architecture, a modified Additive Increase and Multiplicative Decrease (AIMD) parameters are used. Mudassar et al., (2016) noted that for the *cwnd* sizes less than or equal to 38, a similar increase and decrease factors as TCP Tahoe is utilized. Figure 8 demonstrates congestion window growth with time for High-Speed TCP

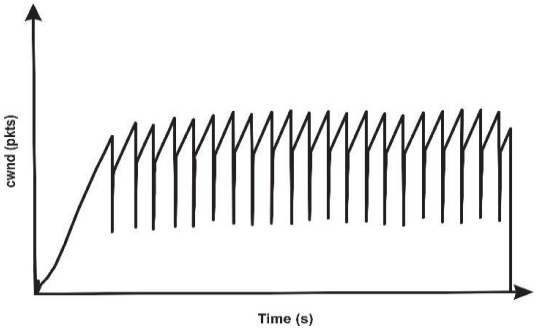

Figure 8: High-Speed TCP *Cwnd* Adjustments against Time

However, when the *cwnd* is more than this value, the function raises the increase factor and reduces the decrease factor proportional to the *cwnd* size. Both the increase factor and the decrease factor are functions of the congestion window:

$$\text{Increase factor} = \frac{F_a(cwnd)}{cwnd} \text{ ; Decrease factor} = g_\beta(cwnd) \quad \text{.....(v)}$$

For the value of *Cwnd* more than or equal to 1 but less or equal to 38, the receiver's congestion window is updated as shown equation (vi) below:

$$\text{Increase: } CWND = CWND + \frac{1}{CWND}$$

$$\text{Decrease: } CWND = CWND - 0.5 * CWND \quad \text{.....(vi)}$$

However, for the receiver congestion windows of more than value of 38, the congestion window is updated as shown in equations (vii) below:

$$\text{Increase: } CWND = CWND + \frac{a(CWND)}{CWND}$$

$$\text{Decrease: } CWND = CWND - b(CWND) * CWND \quad \text{.....(vii)}$$

This means that the value [*a (CWND) / CWND*], represents the increase factor while the value [*b (CWND)* * *CWND*] represent the decrease factor.

D. TCP BIC

In TCP BIC (Binary Increase Congestion Control Algorithm), two window size control policies called additive increase and binary search increase are used to maximize the *cwnd*. Therefore according to Mudassar et al., (2016), for a packet loss, this variant reduces its *cwnd* by a multiplicative decrease factor

β. The *cwnd* size just before reduction is set to W_{max}. However, after reduction *cwnd* is set to $W_{min.}$ Figure 9 gives the representation of these phenomena.

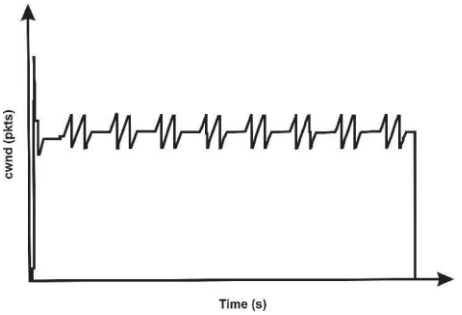

Figure 9: TCP BIC *Cwnd* Adjustments against Time

Given that packet loss happened at $W_{max,}$ the *cwnd* size that the network can presently handle devoid of any loss lies between these two values. Therefore BIC performs a binary search by employing $W_{max,}$ and W_{min} parameters, hopping to the midpoint between these two limits. On condition that the current W_{min} is more than the maximum increment S_{max}, the *cwnd* is enlarged according to the following relation:

$$Cwnd=cwnd + S_{max} \quad \text{.....(viii)}$$

If a packet loss is detected at the new window size, this window size becomes the new W_{min}, else it becomes W_{max}.

E. TCP CUBIC

This TCPP variant implements a new slow-start algorithm, known as the HyStart. This algorithm inhibits long burst losses by locating a safe exit point during slow-start as shown in Figure 10. In effect, this enhances the start-up throughput of the TCP CUBIC for large distance; high bandwidth computer networks (Mudassar et al., 2016).

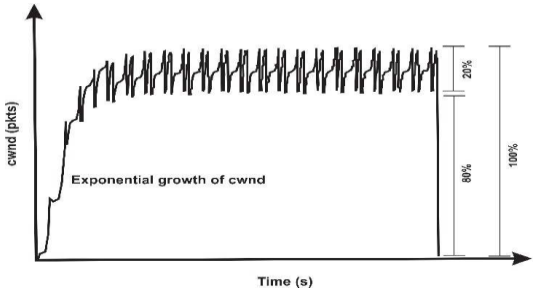

Figure 10: TCP CUBIC *Cwnd* Adjustments against Time

When packet loss occurs, the current *cwnd* is taken to be *Wmax*. Afterwards, the *cwnd* is decremented by a constant decrease factor β and enters into congestion avoidance phase. Thereafter, window size is adjusted upwards until the window size becomes W_{max}.

F. TCP VEGAS

In their study, Afanasyev et al., (2010) illustrates that TCP Vegas utilizes timeouts and round-trip delays measurements for every packet in the transmit buffer. Moreover, an additive increase in the congestion window is employed, resulting in the throughput – time graph shown in Figure 11. This delay-based component is designated, *dwnd* and together with the timeouts, it plays a major role in the determination the value of the new congestion window after a congestion has been detected.

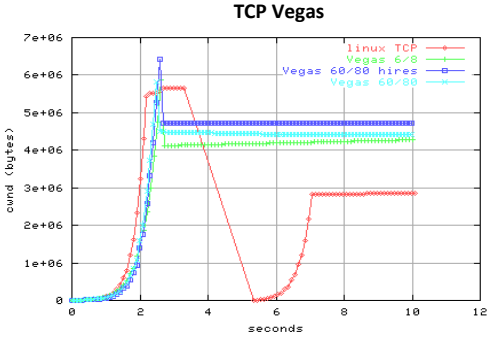

Figure 11: TCP Vegas *Cwnd* Adjustments against Time

G. TCP HYBLA

This approach is ideal for TCP connections that integrate a high-latency terrestrial or satellite radio links. Figure 12 shows the performance of TCP Hybla at different values of round trip time (RTT), where *SS* denotes slow start phase. This figure indicates that on congestion detection, TCP Hybla abandons the slow-start phase and enters the congestion avoidance phase. Shohidul et al., (2009) point out that because of longer round-trip times involved in these links, there is a requirement for an analytical evaluation of the congestion window dynamics, which then necessitates its modifications to eliminate the performance dependence on the round trip times.

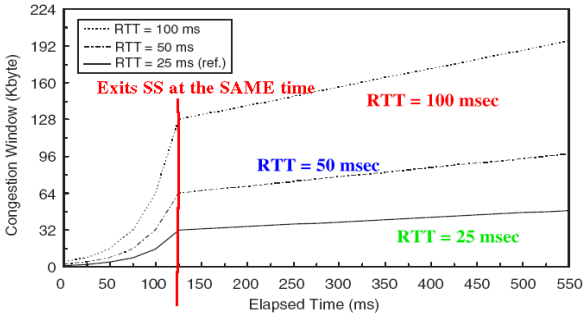

Figure 12: TCP Hybla *Cwnd* Adjustments against Time

H. TCP WESTWOOD +

This variant is grounded on end-to-end bandwidth estimation and it uses three duplicate acknowledgments or a timeouts to detect packet loss as shown in Figure 13. This approximation is employed to set congestion window and slow start threshold after a congestion incident (Afanasyev et al., 2010). The bandwidth approximation involves low-pass filtering of acknowledgment packets returning rate.

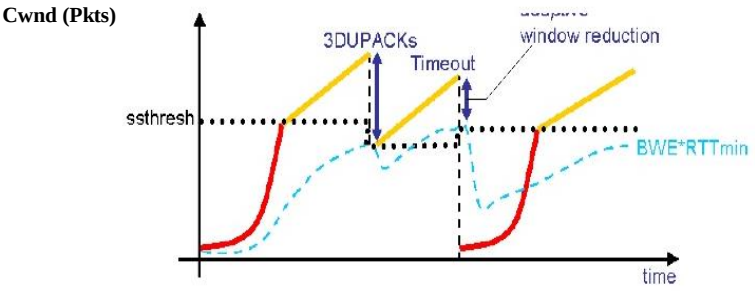

Figure 13: TCP Westwood + *Cwnd* Adjustments against Time

In contrast with TCP Reno, that blindly adjust the new congestion window into half the previous value after three duplicate ACKs, this variant sets a slow start threshold and a congestion window taking into account the bandwidth available at the time congestion is detected. This is illustrated by the relation in Figure 13 ($BWE * RTT_{min}$).

I. COMPOUND TCP

According to Mudassar et al., (2016), this variant was designed for high bandwidth network deployments. It incorporates a scalable delay-based component of TCP Vegas into the standard TCP Reno algorithm. It introduces a new state variable called delay window *dwnd* to current TCP Control Block. The congestion window – time graph of this TCP variant is shown in Figure 14 that follows.

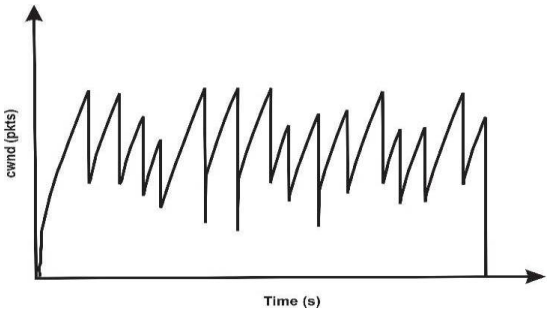

Figure 14: COMPOUND TCP *Cwnd* Adjustments against Time

This new variable is meant to manage the delay based component in compound TCP. During the initial stages of a new connection, this protocol employs the slow-start behavior of the standard TCP by incrementing the congestion window in a similar version to TCP New Reno. However, when the connection enters the congestion avoidance phase, delay-based component is activated. Thus, CTCP maintains two windows concurrently, a regular *cwnd* and a delay window *dwnd* based on delay-based component of TCP Vegas.

In their research work, (Peng, & Low, 2014) noted that TCP performance is anchored on the three main parameters: poor loss detection, coefficients of congestion window (*cwnd*) before or after loss and increase in *cwnd* at the beginning of connection.

It is clear from the analysis of the various TCP variants that these congestion control algorithms all pursue to optimize bandwidth usage by adaptively adjusting the congestion window. Unfortunately all of them use acknowledgements to either detect congestion or to respond to congestions. According to Avi (2016), attacks utilizing these acknowledgements are documented and include session hijacking, sequence number prediction and SYN flooding attack.

V. DISCUSSIONS

The TCP protocols must adequately detect and deal with network congestions if secured data transfers between the various network terminals and the organizational server farms are to be assured. Poor detection of congestion may lead to the re-transmission of earlier packets, which will effectively allow an intruder who has just connected to the network for eavesdropping purposes gather previous communications. Additionally, a malicious user may increase his congestion window, hence effectively increasing his bandwidth at the expense of other TCP users.

This will ultimately lead to low bandwidths for other users, therefore denying them their right to use the transmission media. This then becomes a typical denial of service (DOS) for the affected users. The intruder has now established for himself very large bandwidth that he can utilize for malicious activities, such as transferring large files in a typical advanced persistent threat attack, where large volumes of data exfiltration requires large network bandwidths.

The current TCP variants that implement the four congestion control algorithms have a number of challenges associated with them. To start with, with TCP Tahoe, to detect data packet losses, it takes a complete timeout interval. Another point is that it sends out cumulative acknowledgements, rather than individual acknowledgements for individual data packets. The consequence of this is that each time data packet loss happens; it has to wait for a timeout, a scenario that leads to a high cost in high band-width delay links. TCP Reno's performance is ideal in situations where packet losses are few. It does not have mechanisms for detecting multiple packets loses within one round trip time.

This means that when multiple packet losses are incurred, its performance deteriorate and resembles that of TCP Tahoe. The rationale for this is that in scenarios where there is a manifold packet loss, the first clue concerning this comes when the source receives duplicate acknowledgements. However, indication of the second packet loss will be detected after the acknowledgement of the retransmitted first data segment reaches the sender after one round trip times. The other TCP variants such as Vegas, BIC, CUBIC, compound and Westwood all have fairness issues. This means that they compromise the friendliness concept, which requires that the TCP congestion algorithms adjust very fast in accordance with the network conditions. These algorithms also need to be TCP compatible so that all connections using the additive increase

and multiplicative decrement get their fair share of the network bandwidth.

VI. CONCLUSION

Advanced persistent threats are serious security challenges that focus only on a particular organization or individual. These attacks normally involve six steps. The first step is intelligence gathering, which seeks to identify and research target individuals. This is normally accomplished using public sources such as LinkedIn and Facebook and prepare a customized attack. The second step is the location of the point of entry, which include the establishment of a backdoor that can then be used for created and the network infiltration. The third step is command and control (C&C) communication, which essentially allows the intruder to instruct and control the compromised machines and malware used for all subsequent phases. the fourth step is lateral movement and persistence, where once inside the network, intruder compromises additional machines to gather credentials, escalate privilege levels and maintain persistent control.

The fifth procedure is the asset or data discovery, where several techniques such as ort scanning are used to identify the noteworthy servers and the services that house the data of interest. The last step is data exfiltration, where once sensitive information has been gathered, the data is funneled to an internal staging server where it is chunked, compressed and often encrypted for transmission to external locations. This last step requires very high bandwidth, and is therefore the step where poor congestion handling by various TCP implementations can be employed to increase the intruder's congestion window inappropriately to facilitate faster file transfers from the victim machines.

To this end, this study has yielded fruitful results that could be used to design and implement an adaptive congestion

control algorithm that is bale to address the shortcomings of the current congestion control algorithms. Attacks such as TCP SYN take advantage of TCP's three way handshake to hold network resources at ransom, denying legitimate users opportunities to access them.

The current algorithms for congestion control are also unfair in that network terminals are able to increase their congestion windows and hence their data transfer rates at the expense of other terminals. Moreover, the existing congestion control algorithms are prone to sequence number prediction attacks because after the three way handshake, the sequence numbers and acknowledgement numbers are incremented in a predictable manner. This is an area where intense research needs to be done to address these challenges that are often employed as attack vectors for networked enterprises.

REFERENCES

[1] Avi K., "Lecture 16: TCP/IP Vulnerabilities and DoS Attacks: IP Spoofing, SYN Flooding, and The Shrew DoS Attack", Purdue University, (2016).

[2] Kire J., Slavcho C., Sime A., Lidija G., Oliver I., Leonid D. and Emilija K., "Performance Analysis of Linux-Based TCP Congestion Control Algorithms in VANET Environment", International Journal of Future Generation Communication and Networking, Vol. 9, No. 4, pp. 17-26, (2016).

[3] Christos N. Houmkozlis, George A. Rovithakis , "End-to-End Adaptive Congestion Control in TCP/IP Networks", Artistotle University of Thessaloniki, Greece, (2012).

[4] Jonathan C., "Increasing the TCP initial congestion window", (2012).

[5] Simon J.,and Dimitrios P., "OTCP: SDN-Managed Congestion Control for Data Center Networks", School of Computing Science, University of Glasgow, (2016).

[6] Nishida Y., "The NewReno Modification to TCP's Fast Recovery Algorithm", (2012).

[7] Callegari C., Giordano S., Pagano M. and Pepe T., "Behavior Analysis of TCP Linux Variants", Elsevier Computer Networks, vol. 56, pp. 462-476, (2012).
[8] Dong M., and Schapira M., "PCC: Re-architecting congestion control for consistent high performance," in NSDI 15, (2015).
[9] Stefan S., Cardwell N., Wetherall D., and Anderson T., "TCP Congestion Control with a Misbehaving Receiver", Department of Computer Science and Engineering University of Washington, Seattle, (2015).
[10] Shaneel N., "Improving Network Performance: An Evaluation of TCP/UDP on Networks", Department of Computing UNITEC Institute of Technology Auckland, New Zealand, (2014).
[11] Guang Y., "Introduction to TCP/IP Network Attacks", Department of Computer Science Iowa State University, (2015).
[12] Shohidul I,, Kashem M. Sadid H., Rahman M., Anam S., "TCP Variants and Network Parameters: A Comprehensive Performance Analysis", Proceedings of the International MultiConference of Engineers and Computer Scientists, (2009)
[13] Mudassar A., Asri M., Ngadi, Mohd M., "Experimental Evaluation Of TCP Congestion Control Mechanisms In Short And Long Distance Networks", Journal of Theoretical and Applied Information Technology, (2016).
[14] Robbie M., "Attacks on TCP/IP Protocols", Computer Network Security, (2015)
[15] Bellovin M.,"Security Problems in the TCP/IP Protocol Suite", (2013).
[16] John T., Barry E., "TCP veto: A novel network attack and its application to SCADA protocols", Innovative Smart Grid Technologies (ISGT), IEEE PES, (2013).
[17] Afanasyev, A., Tilley N., Reiher P., KleinrockL., "Host-to-host congestion control for TCP", IEEE Communication Surveys and Tutorials, (2010).
[18] Peng, Q., and S. Low, "Multipath TCP: Analysis, Design, and Implementation", IEEE/ ACM Transactions on Networking, vol. PP, no. 99, (2014).