Volume 3, Issue 10, October 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Secure-BIGWHEEL: A Secure Multi-party Communication Protocol for DDoS Defence Framework in NS2

Anju.K.S[*],Tintu Alphonsa Thomas[†] ,G.S .Santhoshkumar[†]
PG Scholar[*], Assistant Professor[†], Department of Computer Science &Engineering
Amal Jyothi College of Engineering, Kanjirappally, India

Abstract — The general solutions for network-based applications to mitigate DDoS attacks based on port-hopping scheme that enables the applications to mitigate DDoS attacks by changing their communication ports. This solution is motivated by the fact that network-based applications commonly provide some open ports for communication, making they become targets for DDoS attacks. To extend port-hopping to support multi-client applications, by using the Secure- BIGWHEEL algorithm, for each application-server want to communicate with multiple clients in a port-hopping manner, in which there is no need for group synchronization The existing port-hopping to support multiple client applications dynamically changing port numbers during the data transfer. If any one of the port numbers within the interval between the port-hopping is attacked, then the entire communication can be affected. In this situation propose a new secure adaptive multi client algorithm called Secure-BIGWHEEL, in which the data is to be encrypted using the RSA algorithm. The encrypted data are chosen to send from the client to the server. During data transfer client port number dynamically change with respect to the time interval. The encrypted data are transmitted through various port numbers and reaches the server. Even though the dynamic port numbers are known to the attacker data is encrypted form. The same encryption algorithm will be used for both single and multi-party communication. This paper discusses the implementation details about how to simulate a DDoS defence system using Secure- BIGWHEEL algorithm in NS2.

Keywords— Secure-Bigwheel, RSA, Port-hopping, Distributed Denial of Service, Flooding Attacks.

I. INTRODUCTION

A Denial of Service (DoS) attack is any attack that overwhelms website, causing the content normally provided by that website to no longer be available to the visitors of the website. The DDoS attacks are traffic volume-based attacks originating from a set of compromised hosts. These hosts, called as zombies, form a widely distributed attack network known as botnet. Many modern DoS attacks are DDoS attacks. When users feel difficulty in getting to the website and it should not be assumed that the site is under a DDoS attack. Different forms of DoS are far easier to implement than DDoS, and so that the attacks are used by parties with malicious purpose. Many such DoS attacks are easier to defend against once the mechanism used to cause the denial of service is known. Therefore, need to do proper analysis of attack traffic when a site becomes unable to perform its normal function. Today Internet is most common, which is a hostile environment for many attacks. The security measures should ensure the survivability of a system even when facing failures or attacks. The application-level Denial of service attack aims to deplete the resources of end hosts by abusing application traffic. Dealing with such a DoS attack is a challenge that concerns both the industry and the academic community.

Application DoS attacks make under the flaws in the application layer design and the implementation to prevent authorized access to the victims services [3]. They represent a set of attacks on different applications, as they are aimed specifically at disrupting operation rather than diverting the application access controls. Attacks based on exploiting these flaws can over the attacker a number of advantages over traditional DoS attacks:

- **Application DOS attacks are efficient:** The attacker may not need as much resource machines at their disposal to successfully complete the attack. Application level attacks target bottlenecks and resources limitations within the application not use large amount of bandwidth and do not require many compromised zombie systems.
- **The attacks will not be detectable or preventable by existing enterprise security monitoring solutions**: Since the attacks do not consume more amounts of bandwidth and it may be indistinguishable from normal traffic.
- **Application DOS attacks are harder to trace**: Application level attacks normally use HTTP or HTTPS as their transport communication. Many of these proxy servers do not keep logs of connection attempts and could therefore successfully hide the true origin of the attacking host.

A DDoS attacks can be so powerful that they can easily deplete the resources or bandwidth of the potential targets, by flooding massive packets. To solve the threat of the DDoS problem, the internet infrastructures and network applications including services and communication systems for emergency management are under. Many of the network-based applications commonly use some open port(s) for communication, making themselves targets for DoS attacks. Attackers that have the ability of eavesdropping messages exchanged by the application can identify open port numbers and launch

directed attacks to those ports as opposed to blind attacks that can be launched to arbitrary ports, even by non-eavesdropping adversaries. The application parties communicate via ports that change periodically over time, according to a pattern known by both the sender and receiver, such as a pseudo random sequence with common seed- called port-hopping [7].

II. PROPOSED DDOS DEFENCE SYSTEM

Network-based applications commonly open some known communication port(s), making themselves easy targets for (distributed) Denial of Service (DoS) attacks. An attacker, can eavesdrops some packages, discover which port is being used and launch a directed attack over such ports. Taking this scenario in consideration, the solution studied is based on the idea that the parties involved are capable of communicating with each other hopping in between different available ports over time.

A. Proposed System Architecture

An attacker wants to subvert the communication of client-server application by attacking their communication channels or ports. At each time point, some port must be open at the server side to receive the messages sent from legitimate clients. The server side, there are N ports that the server can use for communication. The server and the legitimate clients share a pseudorandom function f to generate the port numbers which will be used in the communication. Using preceding authentication procedure the server can distinguish the messages from the legitimate clients.

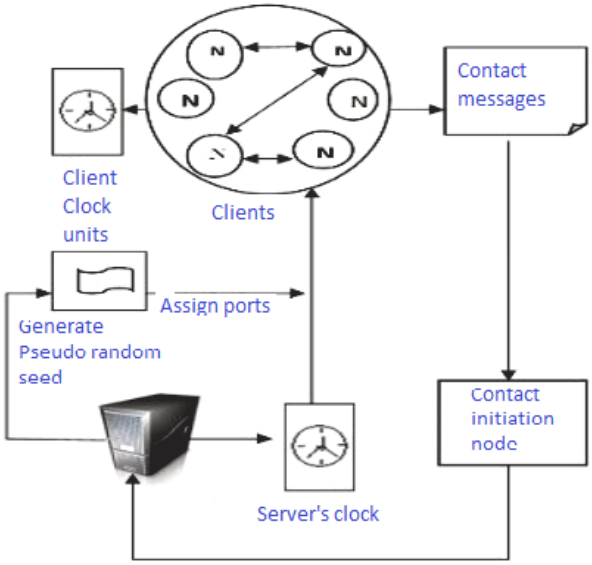


Fig.1 Proposed System Architecture

A Clock drift is used to maintain clock rates between client and server. The clock rates between the client and server adjusted and aligned. The Pseudo random function generates pseudo random seed in the server and it assigns ports to each client. The Client to send contact messages to the server. And the client align the hopping time period at adversary chosen time intervals to control align. Each client adjusts its hopping period length and align its hopping period with the server.

Clients get the seed pseudorandom function from Server to compute the port sequence. The application data is sent from Client to Server which is sent out to the open ports of Server that changes every time units of Server clock, corresponding to client time units in Clients clock. This happens after the contact-initiation part. Periodically, the sender and receiver can use new seeds of the pseudorandom function to generate different port number sequences. This allows the port number sequence which is used for communication is changed periodically [7]. The clients and the server share a pseudorandom function to compute which port should be used in a certain time slot. Every client uses the same pseudorandom function to generate the destination port number.

B. A Port-hopping protocol for Secure Multi-party Communication

In this section the Secure- BIGWHEEL algorithm is considered to deal with multi- party communication, supporting several Clients connected to the same Server. Since each Client follows the Servers hopping procedure, and take the Servers clock as reference, they are capable of communicate independently from each other. The existing BIGWHEEL [2] to support multi-party applications, change port numbers dynamically during the data transfer. Since, there occurs a serious situation that, if any one of the port within the interval is attacked, then the entire communication can be affected. This work propose a new secure adaptive multi client algorithm called Secure-BIGWHEEL [1], in which the data which can be encrypted using the RSA algorithm. The encrypted data is chosen to send from the client to the server. During data transfer client port number automatically change with respect to the time interval. The data is transferred through various ports and reaches the server. This encrypted data cannot be viewed by the attackers even when the dynamic port numbers are tracked. The same encryption algorithm will be used for both single and multi client communication.

The multi-client communication is based on a idea: since each client considers the servers clock as the reference clock, it can interact with the server independently of the other clients. The Secure BIGWHEEL algorithm, aiming at meeting the afore mentioned goals, functions as the Big Wheel rides at amusement parks: clients queue for the next available compartment. Here each compartment represents a hopping sequence; compartments are deployed in a way that aims at balancing the load among them and also at minimizing the clients waiting times to initiate contact with the server.

When using Secure-BIGWHEEL, worker ports still remain open for L + μ units of time but now the Server will support m port number sequences instead of just one this afford more clients and also decrease the maximum waiting time for each one of them. In the Clients side, by usingλ and the pseudo-random function fψ.It is possible to generate different port number sequences if different values of λ are given. Apart from these changes, the phases previously explained and the actions performed in each one of them are the same, so when the server receives a contact-initiation message from the Clients, it will send the reply at the closest opening time of a worker port (considering all m sequences) along with the corresponding value of λ for the sequence to which that worker port belongs. The port hopping mechanism consists of three parts: the contact-initiation part, the data transmission part, and the resynchronization/adjustment part [2].

1) Contact-initiation:

During this phase, the Client contacts the Server without any well- known port being opened at neither the servers' side nor, C having to rely on a third-party to get the port information.

Algorithm for client C in the initiation stage:

T_c = undef ;
Reply=false
Sending contact initiation messages:
While** (Reply ==false) **do
*I=**SELECT** (I_i |iε{1,2,...k})*
for(all ports in I) do
***SEND** (init, time, p)*
end for
***WAIT** (2_ + L)*
end while

Algorithm for server using multiple hopping sequences:

Buffer B stores reply messages that are waiting to be sent.
- receiving contact-initiation message:
***Receive** < **Init, timestamp, key** >*
hc(t_1)$\Leftarrow$ timestamp
t_2 $\Leftarrow$ $Time_{now}$
/ p_j^i is the i^{th} port number in hopping sequence j.*
*Suppose it is open time is the closest to the current time. */*
λ_j $\Leftarrow$ seed for hopping sequence j
$\sigma\Leftarrow$ the corresponding index value for p_j^i in hopping sequence j put the reply message.
*< **ReMsg, λ_j, timestamp, hc (t_1), t_2; key** >into buffer B*
- sending reply messages:
Whenever a new worker port is opened
Send all the reply messages in buffer B to the corresponding clients Clear B

2) Data transmission:

In data transmission C sends encrypted data messages to the worker ports of S. After C gets the reply from the server in the contact-initiation part, C has the seed λ for the pseudorandom function fψ to generate the sequence of the worker ports. The open interval of the worker ports is L + μ time units, where L >μ. The new worker port will be opened μ time units earlier than the closing time of the old one. When S receives the contact-initiation messages from C, it will send the reply message at the time when the next worker port is opened, and the integer σ has the value for generating the next worker port. When C gets the integer σ from Ss reply, it will send the data messages immediately to the port computed from f(λ,σ) C has a timer T_c which will be assigned to 0 when C receives the reply message from S. T_c increases at the same rate as the local clock of C.

Algorithm for client C in Secure data transmission:

$p_{old}\Leftarrow$ fψ(λ,σ)
p_{new} $\Leftarrow$ fψ(λ,σ+ 1)
/P_{old} is the destination port in the current period, and P_{new} is the destination port for the next period. */*

- sending data messages
while has more data to send do
send **<** ***encrptdata,*** P_{old}***,key*** **>**
if (iL-$\mu \leq T_c \leq iL$ *) then*
send< ***encrptdata,*** P_{new}***, key*** **>**
end if
end while
- changing the destination port
If(T_c *= iL)then*
$p_{old} = p_{new}$ *and*
$p_{new} \Leftarrow f\psi(\lambda,\sigma+i+1)$
end if

3).Resynchronization:

This phase is reached after the Client has received the reply from the Server and, before it starts sending data packets; in this phase, the resynchronization algorithm uses the clocks information, from the exchanged messages, to determine whether the Clients clock is slower or faster than the Servers and, based on such, it takes the proper actions to ensure successful data transmission.

receive$(ReMsg, \lambda, \sigma, timestamp, hc(t1), t2)$ in Algorithm 1.
$t_{reply} \leftarrow$the arrival time of ReMsg;
$\rho_u \leftarrow \frac{t_{replay}-T_{send}}{timestamp-T_{arrive}}$
/* where timestamp is th timestamp included in the contact-initiation reply message Remsg */
$\rho_l \leftarrow \frac{t_{reply}-T_{send}}{timestamp-T_{arrive}+2\mu}$
if$(1 \leq \rho_l \leq \rho_u || \rho_l \leq \rho_u \leq 1)$then
$Interval_H \leftarrow \frac{(\rho_u \rho_l \Delta)}{(\rho_u - \rho_l)}$
$if\ (1 \leq \rho_l \leq \rho_u)$ then
$L_c \leftarrow L.\rho_l$
else
$L_c \leftarrow L.\rho_u$
end if
else
$Interval_H \leftarrow min\{\frac{\rho_l}{1-\rho_l}, \frac{\rho_u \Delta}{\rho_u - 1}\}$
end if
Call Algorithm 1 at time $(\text{Time}_{now} + Interval_H)$;

Algorithm for Resynchronization of Multi-party Communication

III. IMPLEMENTATION OF SECURE-BIGWHEEL IN NS2

NETWORK SIMULATOR

Network Simulator 2 is an event- driven simulation tool. It has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols like routing algorithms, TCP, UDP etc can be done using NS2. The NS2 provides users with a way of specifying new and existing network protocols and simulating their corresponding behaviours. The NS2 is widely used in the research community to evaluate protocols in a simulation environment. Although other simulators are widely available.NS2 is free, the source code is available for inspection and modification, and there is a large user community that provides contributions in the form of extensions that allow new protocols and systems to be simulated. That same community has expressed concerns over the process of evolution of NS2, and the impact this might have had on how well NS2 compares to use of protocols in the real world [5].

1) Installing NS2 on UNIX Based System:

1. Download ns-2.35 allinone package and extract ns-allinone-2.35.tar.gz to NS2 installation directory:
   cd /opt
   sudo mkdir ns-2
   cd ns-2
   cp -rp ~/Download/ns-allinone-2.35.tar.gz /opt/ns-2
   tar -zvxf ns-allinone-2.35.tar.gz
   cd ns2/ns-allinone-2.35

2. Install required packages
   sudo apt-get install build-essential autoconf automake libxmu-dev
   sudo ./install
3. Setup environment variable
   vi ~/.bashrc
   ** append the following lines:*
   # LD_LIBRARY_PATH
   OTCL_LIB=/opt/ns-2/ns-allinone-2.35/otcl-1.14
   NS2_LIB=/opt/ns-2/ns-allinone-2.35/lib
   X11_LIB=/usr/X11R6/lib
   USR_LOCAL_LIB=/usr/local/lib
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:
   $OTCL_LIB:$NS2_LIB:$X11_LIB:$USR_LOCAL_LIB
   # TCL_LIBRARY
   TCL_LIB=/opt/ns-2/ns-allinone-2.35/tcl8.5.10/library
   USR_LIB=/usr/lib
   export TCL_LIBRARY=$TCL_LIB:$USR_LIB
   # PATH
   XGRAPH=/opt/ns-2/ns-allinone-2.35/bin:
   /opt/ns-2/ns-allinone-2.35/tcl8.5.10/unix:
   /opt/ns-2/ns-allinone-2.35/tk8.5.10/unix
   NS=/opt/ns-2/ns-allinone-2.35/ns-2.35/
   NAM=/opt/ns-2/ns-allinone-2.35/nam-1.15/
   PATH=$PATH:$XGRAPH:$NS:$NAM
4. Validate ns-2 installation
   cd /opt/ns-2/ns-allinone-2.35/ns-2.35
   ./validate

   It takes around 35 minutes to complete. Make sure that all tests are passed.

2) Adding the new Agent into NS-2:

To create a new type of packet which will include all the information necessary for the BIGWHEEL algorithm (struct hdr bigwheel in bigwheel.h);for this, we defined a new packet type for the bigwheel agent. The rsa agent is created for providing more security in bigwheel agent [6].

Steps to be followed for implementation of new bigwheel and rsa agents are:

1). Paste the bigwheel.h,bigwheel.cc, rsa.h and rsa.cc file in ns-2.35

2). Modify the packet.h under ns-2.35/common directory in the following way

typedef unsigned int packet_t;
.....
.....
// insert new packet types here
static const packet_t PT_BIGWHEEL = 73;
static const packet_t PT_RSA = 74;
static packet_t PT_NTYPE = 75; // This MUST be the LAST one
.....
name_[PT_RAP_DATA] = "rap_data";
name_[PT_RAP_ACK] = "rap_ack";
.....
name_[PT_TFRC]= "tcpFriend";
\textbf{name_[PT_BIGWHEEL]= "bigwheel";
name_[PT_RSA]= "rsa";}
name_[PT_TFRC_ACK]= "tcpFriendCtl";

3. Modify the ns-packet.tcl under ns-2.35/tcl/lib/ directory in the following way

foreach prot {
 # Other:
bigwheel
rsa

4. Modify the ns-default.tcl under ns-2.34/tcl/lib directory in the following way

.....
.....
Agent/bigwheel set packetSize_ 60
Agent/bigwheel set periodL_ 0
Agent/bigwheel set maxDeliveryLatency_ 0
Agent/bigwheel set totalPackets_ 0
Agent/bigwheel set noIntervals_ 0
Agent/bigwheel set cDrift_ 0
Agent/bigwheel set deltaTime_ 0
Agent/bigwheel set noSequences_ 0
Agent/bigwheel set totalPorts_ 0
Agent/bigwheel set sessionStatus_ 0
Agent/rsa set packetSize_ 30

5. Modify the Make_le and Make_le.in in ns-2.35 directory in the following way

*OBJ_CC = *
.....
.....
*apps/pbc.o *
.....
*multiparty/bigwheel.o *
*multiparty/closedPort.o *
*multiparty/rsa.o *
$(OBJ_STL)

6. Go to command prompt and go to the path of ns-2.35 where Makefile stays. And then run**. . /*configure*** and make clean and make command. This will create bigwheel.o and rsa.o fille. That means the NS-2 system will understand the protocol you implemented.

A. Simulation Set-Up

For setting the simulation use NS2 version 2.35 with running Ubuntu 13.04 was used. Ns2 is set to simulate a dumbbell network topology shown in Fig.2.The bottleneck bandwidth of 10Mb/25s is applied at the same time.

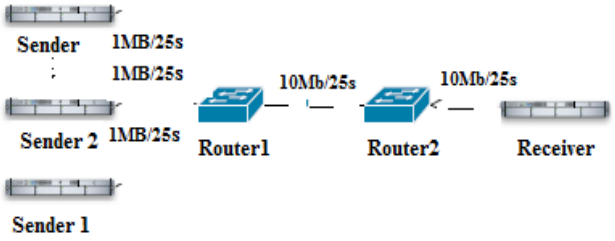


Fig.2 Simulation configuration

For this case, simulation using a three node topology(Fig.3) in which the nodes labeled Client 1 and Central Directory are the ones communicating with each other using new protocol; Gateway node in between, works only as an intermediary in the communication and no bigwheel agent has been attached to it hence, takes no part in the algorithm.

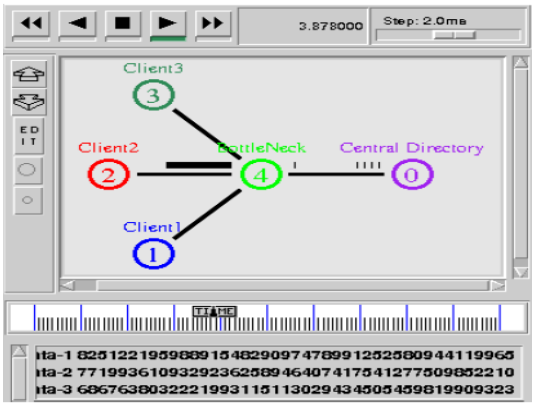
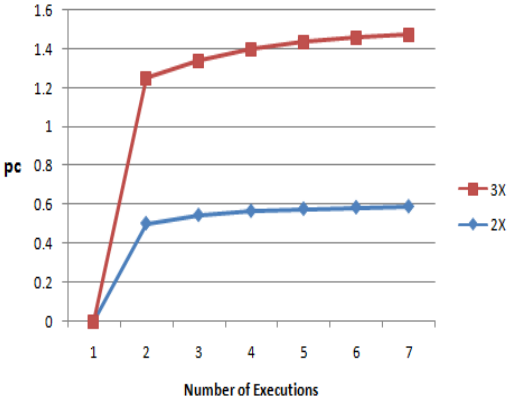


Fig.3 Illustration of Experiment set up of Secure-BIGWHEEL

Table I Parameters obtained from simulation using ρ= 2 and μ = 0.1.

Time	ExecInt	P_c	RhoUp	RhoLow
1	0.62	0.504393	1.68131	2.29933
2	1.22	0.547054	1.82351	2.14226
3	2.12	0.568274	1.89425	2.07958
4	3.32	0.579328	1.93109	2.05014
5	5.12	0.586423	1.95474	2.03225
6	7.82	0.591041	1.97014	2.02102

The value of Pc depending on the case is slowly being adjusted to its ideal value 0.6 or 1.2. Fig.4 shows that the lines are growing dramatically throughout the first times algorithm is executed and, the increment becomes less obvious as they come closer to the values 0.6 and 1.2 correspondingly.



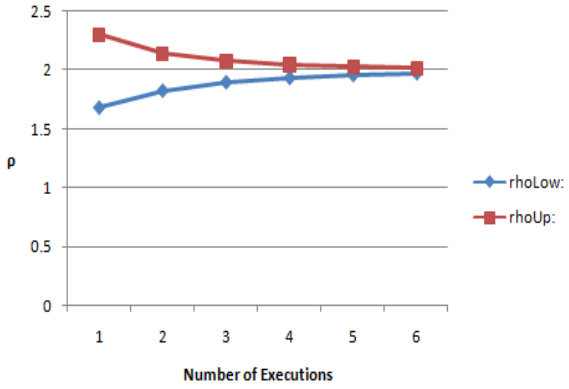


Fig.4 The Pc grow rate for a simulation with ρ=2 and ρ = 3Fig.5 The values of rhoLow and rhoUp simulation with ρ = 2

The rhoLow and rhoUp both remain in between the range $\rho_{Low} \leq \rho \leq \rho_{Up}$ and, effectively approximating to the values defined for each simulation. The Fig.5 shows that the range slowly closes around 2.

B. DDoS Defence Against data Flooding Attacks:

To evaluate the performance of new security protocol, set-up a simulation shown in Fig 5.The data packets are being transferred from the sender to the receiver node throughout router 1 and router 2 however, depending on the scenario, the condition of the network may change affecting the data transfer in between the nodes.

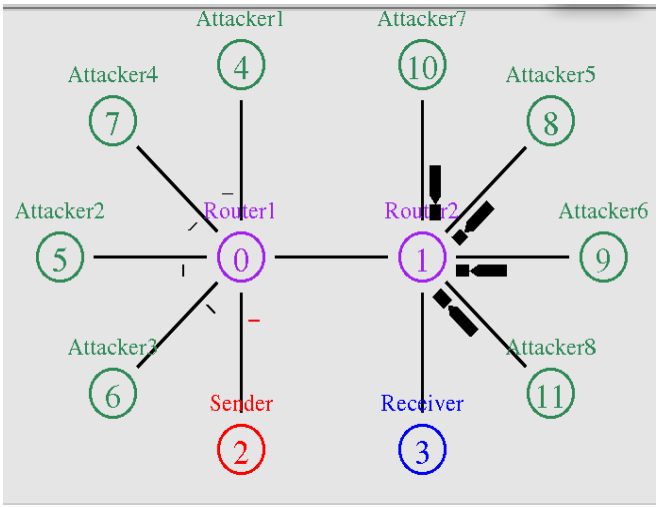


Fig. 5 Illustration of Experiment set up of DDoS Attack

The simulation runs with a receiver and the sender that interchange data packets with each other both with and without the intervention from an attacker. Performance of simulation evaluated with normal TCP network performance with and without new Secure-Bigwheel protocol. Also evaluate the performance with a number of attackers at both the

sender and the reciever. The experiments focus on the metrics of network throughput, end-to-end delay, and the overhead. These metrics are necessary evaluation indexes of the performance of a network. The experiment results from the scenarios of normal operation, the attacked network without proposed defence system and the attacked network with proposed defence system are used to demonstrate the network functionality difference, and the network security performance improved by the proposed defence system.

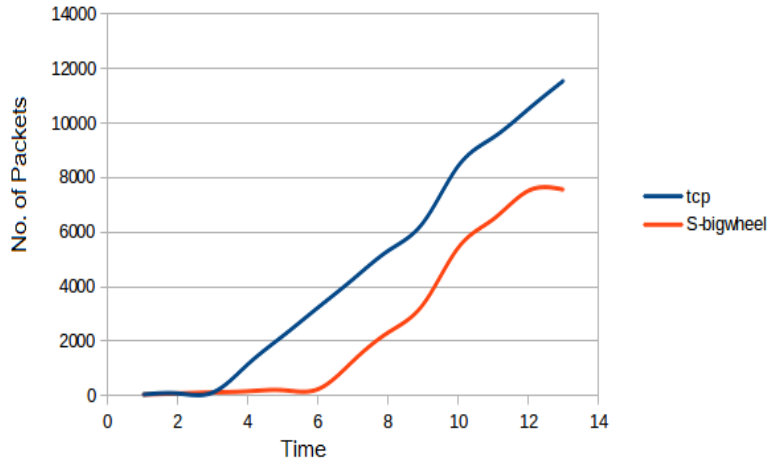


Fig.6 The comparison of Throughput: Tcp and Secure-Bigwheel

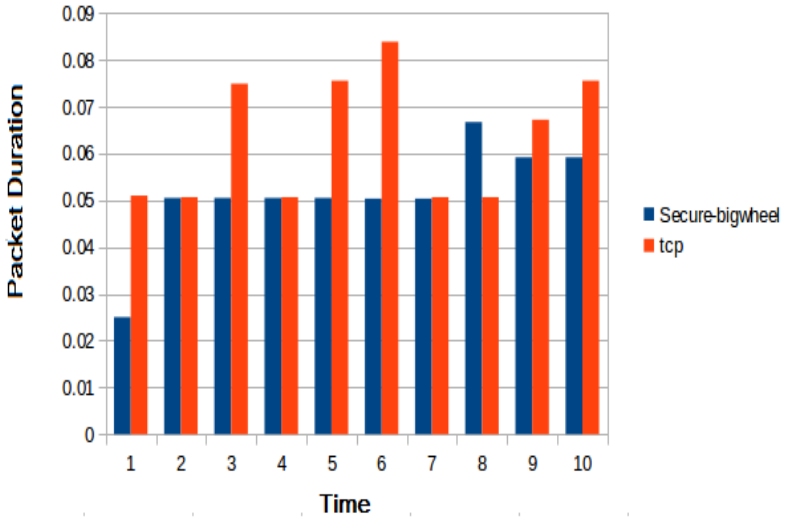


Fig.7 The comparison of End-to-End Delay: Tcp and Secure-Bigwheel

The data packets are transmitted from the sender to the receiver node throughout router 1 and router 2. The DDoS defence system helps the network gain more useful throughput under DDoS data flooding attacks (Fig. 6).The overhead derived from the Secure-Bigwheel implementation is reduced and the amount of data packets being sent overcome the amount of contact-initialization packets required .But the amount of data packets to send is too small then, the cost due to the defence framework, would be relatively larger. The algorithms overhead will be balanced by the amount of data to be transmitted and the reliability of the network whenever the latter is under a DDOS attack. The DDoS defence system may take longer processing delay time and retransmission attempts to send some certain packets successfully, but gain higher overall delivery rate.

IV. CONCLUSIONS

This work, investigate application-level protection against DDoS attacks. An algorithm is presented for a server to support port hopping with many clients. The new secure adaptive multi client algorithm called Secure-BIGWHEEL, provide more secure communication with little overhead.

- The new multi-party port hopping protocol support multiparty applications in a scalable way.
- The proposed method can work under timing uncertainty and specifically fixed clock drifts.

- This scheme does not require any change to existing protocols, nor does it require any new addition protocol.
- It does not require any changes to be made in the Internet infrastructure.
- To facilitating the deployment of this scheme, which will enable legitimate requests from trusted clients to access the server even when it is under severe DoS/DDoS attacks.
- The secure-Bigwheel protocol support more secure multi-party communication with little overhead.

REFERENCES

[1] Anju. K..S, Tintu Alphonsa Thomas, G. S. Santhoshkumar, A Dynamic Cryptographic Approach To Defend Against Distributed DoS Attacks In Multiparty Applications, *International Journal of Engineering Research Technology (IJERT),* ISSN: 2278-0181,Vol. 2 Issue 8, August – 2013

[2] Z. Fu, M. Papatriantafilou, and P. Tsigas ,"Mitigating Distributed Denial of Service Attacks in Multiparty Applications in the Presence of Clock Drifts", *IEEE Transactins on Dependable and secure computing*, Vol. 9,No. 3, 2012

[3] T. Siva, E. S. Phalguna Krishna, K. Pavan Kumar, Preventing ADDOS Attack by Using Secure TRNG Based Port Hopping, *American Journal of Engineering Research (AJER)* ,Volume(Issue-05) , pp-194-199,2013

[4] G. Badishi, A. Herzberg and I .Keidar, "Keeping Denial-of-Service Attackers in the Dark", *IEEE Trans. Dependable and Secure Computing*, vol.4,no.3,pp. 191-204, July-Sept.2007.

[5] Teerawat Issariyakul, Ekram Hossain, *Introduction to Network Simulator,*NS2-springer,2009

[6] Greis M, *Tutorial for the network simulator Ns*, Retrieved September 30, 2009 from :http://www.isi.edu/nsnam/ns/tutorial

[7] Zhang F U, "Multifaceted Defence Against Distributed Denial of Service Attacks: Prevention, Detection, Mitigation", Ph.D. Thesis, Division of Networks and Systems, Chalmers University, 2012

