

PETRI NET MODELING OF COMPUTER VIRUS LIFE CYCLE

Ikekeonwu, G A. M and Bakpo, F.S.
Department of Computer Science
University of Nigeria, Nsukka

ABSTRACT

Virus life cycle, which refers to the stages of development of a computer virus, is presented as a suitable area for the application of Petri nets. Petri nets a powerful modeling tool in the field of dynamic system analysis is applied to model the virus life cycle. Simulation of the derived model is also presented. The intention of this paper essentially is to show that similar procedure can be used to derive anti-viral programs based on the Petri net framework.

Keywords: Virus lifecycle, Petri nets, modeling. simulation.

1.0 INTRODUCTION

The Oxford Advanced Learner's Dictionary defines virus as "a hidden code within a computer program intended to cause errors and destroy stored information⁵". In², a computer virus is defined as a code that attaches itself to other programs in order to alter their behavior, often in a harmful way. Computer virus infection happens to be the most common computer security problem in the computer community today because it poses a serious threat to the computer users. Problems arise from viral infection have imposed significant amount of financial losses to individuals as well as to corporate organizations. The computer virus problem was first described in 1984, when the results of several experiments and substantial theoretical work showed that:

- (i) Viruses could spread essentially unhindered even in the most secured computer systems;
- (ii) They could cause widespread and essentially unlimited damage with little effort on the part of the virus writer;
- (iii) Detection of viruses was undecided;
- (iv) Many of the defenses that could be devised in relatively short order were

ineffective against a serious attacker; and

The best defenses were limited because of inadequate information flow and limited sharing.

The effect of viruses can range from mildly annoying (for example, interruption with a message that appears periodically on the screen) to devastating (for example, an inaccessible hard disk, formatting hard drive, destroying BIOS files). The smallest virus code is just enough to destroy or corrupt data on any computer system. Like its biological counterpart, a virus cannot operate on its own and must enter a host program in order to be activated. Some viruses run immediately, while others wait for a specific date or event. When it runs it copies itself into another program, document or boot sector on the computer. A virus usually executes when a user opens an infected document, runs an infected file or boots from a disk with an infected boot sector. This phenomenon is also true with computer worms programs that copy themselves to another computer or drive usually by means of a computer network. E-

mail worms spread fast because they mail themselves to every entry in the address book on each computer they invade. Worms are often disguised as images, joke software or anti-virus software. We have seen that the development of new technologies such as the Internet, which have made it easier for us to communicate with one another as in a global village, had also provided yet further ways in which virus writers launched their attacks on remotely interconnected systems. Thus we refer to computer virus to mean both computer viruses and worms, since both can be propagated either ways. Examples of viruses include Melissa (1999), Loveletter (2000), W32/Sircam (2001), etc

2 The virus life cycle

The existence of a computer virus, like its biological counterpart in a host machine typically encompasses four stages⁶, namely:

- (i) Dormancy;
- (ii) Propagation (self-propagation);
- (iii) Triggering and
- (iv) Damage

Figure 1 depicts the life cycle

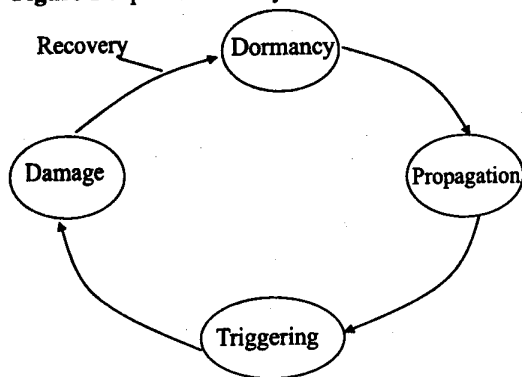


Fig. 1: A virus life cycle

(i) **Dormancy:** Upon infecting a new machine or a new program, the computer virus may remain dormant for a while to avert suspicion. The duration of the dormant period may vary depending on the type of mechanism being used. Some viruses may await a certain number of executions of the

lost program or the elapsing of a certain period of time before progressing to the next stage in the cycle.

(ii) **Propagation:** In the propagation stage, he virus attempts to send copies to other programs on the host machine or machines in the network. The target here may be the COM-file, EXE-file, BOOT, partition or data. A virus may spread by searching the system (disks or server) for uninfected programs and attaching itself to them. In ³, virus propagation mechanisms, are categorized as one-to-one, one-to-many, many-to-one, and many-to-many. Depending on the particular mechanism in use, the viral code modifies an executable file in such a manner that it receives control upon program activation. When an infected program is executed, the virus code is propagated to the target and sets its stage to dormancy. Control is then usually returned to the host program for normal operations. In this way, the virus can hide its existence all the way through the triggering stage.

(iii) **Triggering stage:** This stage facilitates transition to damage stage. The trigger itself may be defined as either one of the following:

- (i) A count of replications: To cause the damage at the count down of the number of copies already generated;
- (ii) A timer- based: To cause the damage at some specific time or date;
- (iii) An event-driven: To cause the damage at soon as a user clicks on a folder or a hyperlink, etc.

(iv) **Damage stage:** The damage itself can range from mildly annoying to malicious, e.g. loss of data, program failure such as inability to load, failure to boot, loss of man hours and anxiety. Nearly all viruses require some stages of development, watch over system and user's activities and exhibit certain behavior. Petri net offers a viable modeling tool for implementation of virus life cycle. Areas of successful applications of Petri nets include distributed database systems, communication protocols, performance evaluations of complex systems, workflow managements, game systems etc. For an in-dept review of Petri nets the reader is referred to ^{3, 7,8, and 11}.

3. Petri nets theory

The theory of Petri nets has its origin in Carl Adam Petri's dissertation "Kommunikation mit Automaten", submitted in 1962 to the Faculty of Mathematics and Physics at the Technische Universität Darmstadt, Germany⁹. It is therefore not surprising from where this vibrant theory derives its name: their inventor **Prof. Carl Adam Petri**. The Petri nets group defines Petri nets as a formal, graphical, executable technique for the specification and analysis of concurrent, discrete event dynamic systems- a technique currently undergoing standardizations. It is also a modeling language', Formally, a Petri net \mathcal{N} is a triple $\mathcal{N} = (P, T, A)$ where:

P is a finite, nonempty set of places (graphically represented as circles);

T is a finite, nonempty set of transitions (graphically represented as bars /);

A is a set of directed arcs, which connect places with transitions and transitions with places. (t, P_i) (P_j, t) (t, P_j) (P_i, t) (t, P_j) (P_i, t)

This expression is interpreted as follows" for all transitions in the net there exist input and output places such that the flow from input place to transition and from transition to output place is indicated by set of directed arcs. The sets of all input and output places of a transition t are denoted by $\text{Inp}(t)$ and $\text{Out}(t)$, respectively. Similarly, the sets of input and output transitions of a place P are denoted by $\text{Inp}(P)$ and $\text{Out}(P)$. A place P is shared iff it is an input place for more than one transition. A net is conflict-free iff it does not contain shared places. Only conflict-free Petri nets are considered in this paper. To be useful tokens or dots are usually assigned to some places in a Petri net setting. Such Petri nets with tokens are called marked Petri nets. A marked Petri net is a pair $M = (\mathcal{N}, m)$ where: \mathcal{N} is a Petri net $\mathcal{N} = (P, T, A)$,

m is an initial marking function which assigns a nonnegative integer number of tokens to each place of the net; $m: P \rightarrow (0, 1, \dots)$.

A transition t is enabled by a marking m iff every input places of this transition contains at least one token. The set of all transitions enabled by a marking m is denoted by $E(m)$. Every transition enabled by a marking m can fire. When a transition fires, a token is removed from each of its input places and a token is added to each of its output places. This determines a new marking in a net and implies a new set of enables transitions, and so on. Ordinarily, a marked Petri net is still not enough for the study of systems performance since no assumption is made on the duration of systems activities. Ramchandani¹⁰ has introduced the timed Petri nets by assigning firing times to the transitions of Petri nets. Molloy⁷ introduced stochastic Petri nets in which transition firing times are exponentially distributed random variables, and the corresponding firing rates are assigned to transitions of a net. This paper dwells on M-timed Petri nets (or stochastic Petri nets). An M-timed Petri net T is defined in 7.12 as a pair $T = (M, r)$, where: M is a marked Petri net; $M = (\mathcal{N}, m)$, $\mathcal{N} = (P, T, A)$,

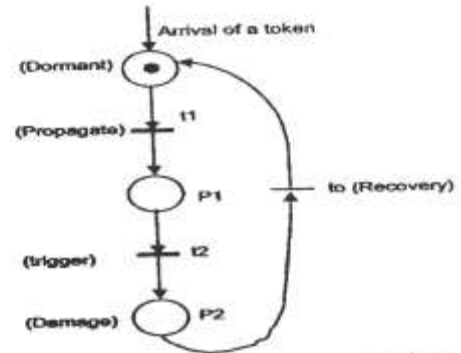
r is a firing rate function which assigns a positive real number $r(t)$ to each transition t of the net. $r: T \rightarrow \mathbb{R}^+$, where \mathbb{R}^+ denotes the set of positive real numbers. The firing time of a transition t is a random variable $v(t)$ with the distribution function: $\text{Prob}(v(t) > x) = e^{-r(t)x}$, $x > 0$.

4. Petri net Modeling and Simulation of a virus life-cycle

4.1 Modeling

We choose to model this system the virus life-cycle using Petri nets because given that virus stages of development is complex and its self-propagation method inherently distributed, we want both to establish that the system behaves correctly and to ensure that its performance is acceptable. Ideally, we would prefer a model that can capture the logical behavior (i.e. interactions that

influence the information flow) of the system, but which could also be extended so as to analyze system performance. The traditional performance models such as queuing network models, finite state machines, turning machines etc, cannot capture important characteristics of this system like concurrency, synchronization or conflict of actions, which are important for the study of their correctness. Petri nets are one such model, which bring all these distinctly different characteristics into a single and elegant framework of representation. The underlying graphical background facilitates visualization of complex process. Figure 2 shows the Petri nets model of a computer virus life cycle.



2 (a) Schematic of a one-to-one propagation

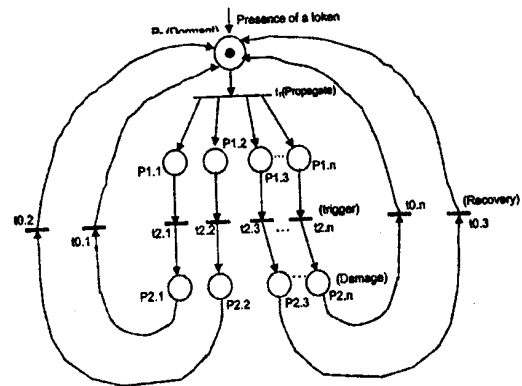


Fig 2(b) Schematic of One-to-many relationship

Net symbols	Descriptions
.	Is a dot. In Petri net configuration, a dot refers to a token (marking) and in our model it shows the presence of a virus.
P_0	Is an initial place in the net. P_0 refers to the dormant stage (condition) in the life cycle.
t_1	Is a transition whose firing propensity depends on the availability of at least a token in place (P_0). t_1 stands for the propagation event.
$P_{1.1}, P_{1.2}, \& P_{1.n}$	Number of places that will receive a token following firing of transition t_1 These refer to the generated copies of the same virus, where n is the number of copies (folders) infected in a given drive. As each token is deposited in places $P_{1.1}, P_{1.2}, \& P_{1.n}$ that it sets its phase to dormant, however the presence of a token soon enables the firing of corresponding output transitions.
$t_{2.1}, t_{2.2}, \& t_{2.n}$	Refers to the triggering event. In this model our triggering event is the countdown of number of replicated copies of the virus.
$P_{2.1}, P_{2.2}, \& P_{2.n}$	Stands for the full brown damage condition of the system. As said earlier the damage done to a system varies. Our model implements a mild form of damage- a message pay your bill is displayed.
$t_{0.1}, t_{0.2}, \& t_{0.n}$	This refers to the recovery event or method applied in order to recover from the damaged condition created in places $P_{2.1}, P_{2.2}, \& P_{2.n}$ An infected system usually remains in the damage state until an up-to-date anti viral software is applied. If the virus scanner successfully detects and cleans the virus, normalcy is said to be restored in the system.

The availability of at least a token (virus) in place (Po) will enable the firing of the transition (t_i), which propagates copies to the entire available places link with Po (addresses or folders). This in turn sets the pace for a self-contained or in-house propagation, leading to a full-brown find folders; damage situation. This process is analogous to a recursive algorithm or recurrence dormant equation a technique for performing task T by performing a similar task T, The task T, is exactly the same in nature as the original task T; however, it represents a solution to a smaller problem than T.

4.2 Simulation

In this section we considered the simulation of a virus life cycle by programming (pseudo-code) in VB. Real life viruses execute arbitrary code once they have replicated themselves on a new host. The only way to simulate this generality is to allow simulated virus the same ability. To properly capture the logical behavior of a virus life cycle, figure 3 depicts the system pseudo-code.

(i) Virus life- cycle pseudocode

Module virus: {inserts itself in another folder}
Get Drive_label;

```

Stage$ = dormant
Procedure s_dormant;
begin
  evaluate dormant-waiting condition;
  if done-waiting then stage$ := propagate
end {s_dormant}
Procedure s_propagate;
begin
  find folders;
  Copy virus_Definition to folder and set stage$ =
  dormant
  if propagate_done then stage$ = trigger
end; {s_propagate}
Procedure s_trigger;
begin
  evaluate trigger condition;
  if trigger:= on then stage:= damage
end; {s_trigger}
Procedure s_damage;
begin
  do damage
end; {s_damage}
begin; {control initially transfers here}
  if stage$:= dormant then s_dormant
  else if stage$:= Propagate then s_propagate
  else if stage$:= trigger then s_trigger
  else s_damage;
goto beginning of the host program

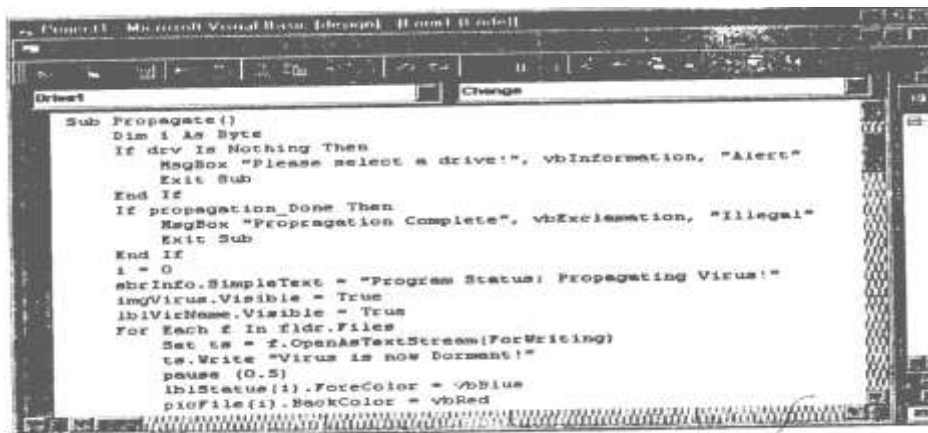
```

Fig 3 Pseudocode of a computer virus life cycle

The actual implementation in visual BASIC is presented in figure 4a and 4b, respectively.

Fig 3 Pseudocode of a computer virus life cycle

The actual implementation in visual BASIC is presented in figure 4a and 4b, respectively.



```

Sub Propagate()
  Dim i As Byte
  If drv Is Nothing Then
    MsgBox "Please select a drive:", vbInformation, "Alert"
    Exit Sub
  End If
  If propagation_Done Then
    MsgBox "Propagation Complete", vbExclamation, "Illegal"
    Exit Sub
  End If
  i = 0
  shrInfo.SimpleText = "Program Status: Propagating Virus!"
  imgVirus.Visible = True
  lblVirusName.Visible = True
  For Each f In fldr.Files
    Set ts = f.OpenAsTextStream(ForWriting)
    ts.Write "Virus is now dormant!"
    pause (0.5)
    lblStatus(1).ForeColor = vbBlue
    picFile(1).BackColor = vbRed
  Next f
End Sub

```

Fig. 4(a) Sample VB codes for virus life cycle simulation

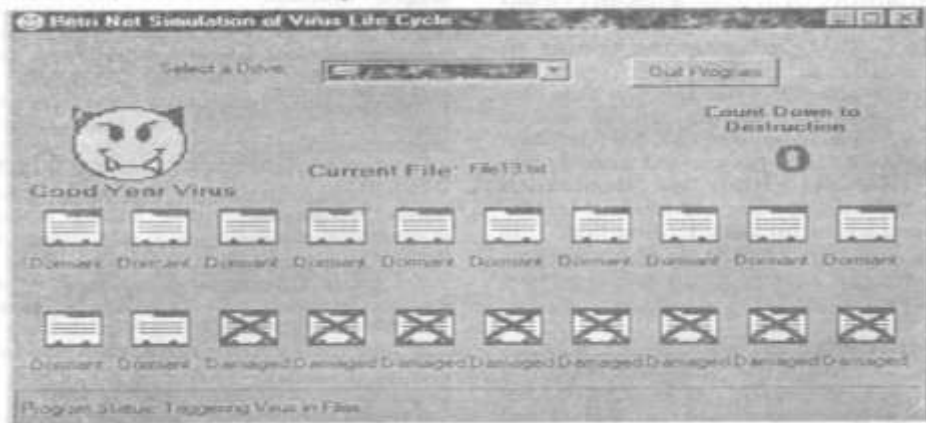


Fig. 4(b): The simulation interface for virus life cycle

In this design, we simulated a virus called Good Year. As the user selects a drive by an event 'click' (i.e., a token is present), the virus propagates by assigning copies to each of the folders/files in the drive and sets its development stage to dormant. It then begins verifying if propagation was successful by counting down the files from inherently a stochastic Petri net

distribution with $v(t) = e^{rt}$. In practice, the number of infected components ¹¹, $N(t) = e^{(g-a)t}$, where t is the elapsed time; g - is the generation rate (the average number of infected system); a - is the absorption rate (the number of worms that die out in an infection cycle and will not cause any further infections); and $-$ is the

System	T(s)	g(ms)	a(ms)	σ(MHZ)
1	0-4	2	0.1	400
2	0-4	1.5	0.1	500
3	0-4	1	0.1	600
4	0-4	0.5	0.1	700
5	0-4	0.12	0.1	800

The complete source code is available on request.

maximum to zero or destruction. The count down process is actually the triggering stage and is the forerunner of the damage stage. At damage, each infected file is merely increased in byte sizes. The entire process is cycle time of an elementary virus spread. We tested this equation by creating Matlab files for five different computer systems according to the data shown in table 2 (below).

4 Conclusion

To increase the body of knowledge on the

subject matter, we offered an in-depth examination of a virus life cycle. This application domain is presented in such a way that Petri nets tool could easily be proffered. Petri nets concept is then applied to model virus life cycle, which in turn is simulated qualitatively. This paper reinforces the validity and efficiency of Petri nets as a modeling tools and advances the current understanding of both virus life cycle and the use of Petri nets theory for modeling and simulation of intricate, life critical processes.

REFERENCES

1. Bakpo, F.S. (2003) "Software Design Modeling with functional Petri Net" *Nigeria Journal Technology Vol. 22, No. 1, March 2003, 15-22.*
2. Bakpo, F.S. (2002) *Computer Use and Applications* Godjiksos Publishers, Nsukka, 128 pp.
3. Chen, L. and Carley, K.M. (2002) "A Computational Model of Computer virus propagation".
http://www.Virusbtn.Com/old/other_papers/Good_vir/
4. Cohen, F. (2004) "Trends In Computer Virus Research" <http://vx.netlux.org/lib/afco6.html>
5. Hornby, A.S. (1995) *Oxford Advanced Learner's Dictionary of Current English* (5th ed.), Oxford University Press, Great Britain.
6. Milenkovic, M. (1982) "Operating Systems: Concepts and Design". *McGraw-Hill Computer Science Series*, Singapore, 755pp.
7. Molloy, M.K. (1982) "Performance Analysis using stochastic Petri nets", *IEEE Trans. On Computers* Vol. 31(9), PP. 913-917.
8. Parthasarathy, S. (2004) "A one-day Tutorial on Petri nets".
<http://www.algolog.tripod.com/petritut.htm>
9. Petri, C.A. (1962) "Kommunikation mit Automaten". Ph.D. thesis, Institute for instrumentelle Mathematik, Bonn [in Germany]
10. Ramchandani, C. (1974) "Analysis of Asynchronous Concurrent Systems by Timed Petri Nets; *Project MAC Technical Report, MACTR-120*, Massachusetts institute of Technology, Cambridge, MA.
11. Szappanos, G. (2000) "Models of Viral Propagation"
<http://www.securityfocus.com/infocus/1265>
12. Zuberek, W.M. (1985) "Performance Evaluation of Concurrent Systems using Timed Petri Nets." Proceedings of the 1985 ACM Computer Science . Conference Agenda for Computing Research, pp. 326-329.