Bösartiges Verhalten von Teilnehmern in durch ns-2 simulierten Netzwerken

Alexander Küchler
Hasso-Plattner-Insitut, Universität Potsdam
Email: alexander.kuechler@hpi.uni-potsdam.de

Abstract

In diesem Dokument werden mögliche Definition von Bösartigkeit angesprochen und mögliche Implementierungen der Bösartigkeit in dem Programm ns-2 zur Simulation von Netzwerken aufgezeigt. Dazu werden grundlegende Gedanken zur Beschreibung von bösartigem Verhalten präsentiert. Anschließend werden verschiedene Implementierungen in ns-2 am Beispiel der DelayBox, des ErrorModel und einer Implementierung im Routing-Protokoll vorgestellt. Die Simulation der Bösartigkeit bildet die Voraussetzung für die Implementierung neuer Feedbackmechanismen und ist somit essentiell für die Überprüfung der Funktionstüchtigkeit.

I. Beschreibung von Bösartigkeit aus verschiedenen Bereichen

Der Begriff der Bösartigkeit kann in verschiedenen Bereichen andere Ausprägungen haben. Deshalb wird zunächst das Verständnis des Begriffs ”Bösartigkeit” aus Bereichen wie Medizin, Philosophie, Rechtssprechung und Soziologie im Ansatz beschrieben, um anschließend für den vorliegenden Anwendungsbereich der Informationstechnologie definiert zu werden.

In der Medizin wird ”bösartig” als Klassifizierungsmerkmal von Geschwüren verwendet. Der zugrunde liegende Entstehungsgrund im Sinne einer unerwarteten Zellteilung und somit rapide steigendem Zellwachstum sowie dessen Verhalten werden dabei beurteilt. Bösartige Geschwüre werden zumeist als Krebs bezeichnet und bilden eine der häufigsten Todesursachen in Deutschland.

In der Philosophie wird der Begriff der Bösartigkeit bei der Einschätzung von Verhaltensmustern verwendet. Eine Person wird nach ihrem Verhalten und der Wahrnehmung durch andere Personen in Bezug auf dieses Verhalten bewertet. Wird das Verhalten als schädlich eingestuft und wird eine zugrunde liegende Absicht unterstellt, ist es ein bösartiges Verhalten. Hier spielen zahlreiche Wahrnehmungen eine Rolle: die von der Aktion betroffenen Person gegenüber der agierenden Person, die der agierenden Person bezüglich ihres eigenen Verhaltens und die mögliche Einschätzung einer nicht von der Aktion betroffenen Person. Insbesondere das Bewusstsein über die Auswirkungen des Handelns spielt hier eine besondere Rolle.

In der Rechtssprechung wird Bösartigkeit zumeist mit Vorsatz oder Mutwilligkeit assoziiert. Jemand der in vollem Bewusstsein über die Auswirkungen seines Handelns ausübt, was anderen schadet, handelt demnach vorsätzlich. Danach beinhaltet ein solcher Vorsatz sowohl das Wissen als auch das Wollen der Tatbestandsverwirklichung zum Zeitpunkt der Begehung der Tat [1].

Im sozialen Umfeld eines Menschen kann sein Handeln als bösartig eingestuft werden, wenn es von der durch das soziale Umfeld explizit oder implizit definierten Norm abweicht. Dabei ist es unerheblich, ob dieses Verhalten in einem anderen Umfeld anders behandelt werden würde. Es spielt lediglich eine Rolle, was das direkte soziale Umfeld für ein Verhalten erwartet. Dieses erwartete Verhalten wird zumeist als erstrebenswertes Verhalten oder Moral bezeichnet.

II. Definition von Bösartigkeit in der IT

Was aber macht nun Bösartigkeit in der Informationstechnologie aus? Und was genau ist bösartiges Verhalten eines Teilnehmers in einem IT-Netzwerk? Als Teilnehmer wird hier lediglich eine Maschine angesehen; es geht hier ausdrücklich nicht um menschliche Teilnehmer an den Netzwerken. Eine solche Maschine ist zunächst einmal programmiert, das bedeutet ihr Verhalten ist vollständig definiert. Davon ausgenommen sind somit neuronale Netze oder maschinelle Teilnehmer mit künstlicher Intelligenz, die durch Sammeln von historischen Daten Verhaltensmuster ändern können. Dies bedeutet, dass das Verhalten zu 100% bekannt und vorhersagbar ist, wenn keine Fehler in der Programmierung vorliegen. Spontanes oder selektives Verhalten, was sich durch eine unterschiedliche Aktion bei gleichen Voraussetzungen auszeichnet, ist somit ausgeschlossen. Die Bösartigkeit ist jedoch nicht nur im Handeln selbst, sondern auch im Ergebnis zu erkennen. Das bedeutet, dass eine Aktion selbst noch nicht bösartig sein muss, aber das Ergebnis der Aktion eben bösartig ist, weil es nicht das gewünschte Ergebnis darstellt. Das Löschen eines Paketes z.B. stellt für sich allein genommen kein bösartiges Verhalten dar. Im Gegenteil: das Löschen eines fehlerhaften Paketes ist u.U. sogar ein erwünschtes Verhalten. Das Löschen eines nicht fehlerhaften Paketes, welches weitergeleitet werden sollte, ist jedoch als bösartig anzusehen. Der Kontext der Handlung selbst spielt also ebenso eine Rolle.

Welche Gründe können nun jedoch zu Änderungen im definierten (programmierten) Verhalten führen? Die folgenden Gründe geben eine erste Übersicht, die unter keinen Umständen Anspruch auf Vollständigkeit erhebt:

- vorsätzlich vorgenommene Veränderungen am definiertem Verhalten durch andere Teilnehmer, z.B. Umprogrammierung durch Angreifer
- auftretende Umweltveränderungen, die Voraussetzung für die volle Funktionstauglichkeit bilden, z.B. das Auftreten von Spannungsschwankungen oder Sensorirritationen
- Erreichen nicht definierter Zustände durch mangelhafte Programmierung
- Selbstschutz als Teil definierten Verhaltens, z.B. Funktionseinschränkungen bei Fehlern in der Selbstdiagnose oder bei zu geringen Energiereserven

Da jedoch nicht die Untersuchung der Gründe hier im Vordergrund steht, sondern eine Implementierung der möglichen Folgen der Bösartigkeit, wird auf die genannten Gründe nicht weiter eingegangen.

Aus den bisherigen Beschreibungen soll nun eine passende Definition für die Bösartigkeit von maschinellen Teilnehmern in IT-Netzwerken gefunden werden:

"Bösartigkeit ist ein nicht selektives, messbares Verhalten, welches eine Abweichung ggü. dem Verhalten anderer Teilnehmer oder dem eigenen definiertem Verhalten darstellt. Dieses messbare Verhalten überschreitet einen Schwellwert, der eine Klassifizierung der Bösartigkeit ermöglicht."

III. Simulation von Bösartigkeit

Für die nachfolgenden Simulationen von Netzwerken und ihren Teilnehmern, soll lediglich die Auswirkung des bösartigen Verhaltens weiter betrachtet werden. Dies wird auf das Senden und Empfangen von Paketen weiter eingeschränkt. Die folgenden messbaren Abweichungen werden dabei berücksichtigt:

- Nicht-Senden von Paketen (*loss*)
- verzögertes Senden von Paketen (*delay*)
- Mehrfach-Senden von Paketen (*reply*)
- Verändern von Paketen bzgl. Inhalt und Metadaten wie Empfänger (*modify*)
- Hinzufügen von Paketen (*add*)

Bei all diesen Aktionen ist davon auszugehen, dass es sich im jeweiligen Fall um ein nicht erwünschtes Verhalten handelt. Um dieses Verhalten nachzubilden, wird ein Netzwerk mit seinen Teilnehmern durch Software simuliert. Als Software kommt dabei ns-2 [2] zum Einsatz. Das Ziel ist die spätere Implementierung der Feedbackmechanismen aus [3]. Dazu müssen eben diese Abweichungen erzeugt werden, um die Nutzbarkeit der Feedbackmechanismen einzuschätzen.

IV. Netzwerksimulation mit ns-2

ns-2 ist eine open-source Software für Netzwerksimulationen. Sie arbeitet eventbasiert und ist single-threaded. Dabei werden der Entwurf sowie Protokolle von kabelgebundenen und kabellosen Netzwerken unterstützt. Die Programmierung geschieht durch die *Tool Command Language* (TCL) bzw. oTCL (object TCL), welches als einfache Skriptsprache wiederum auf Implementierungen in C/C++ zugreift. Zur einfachen Erstellung von Netzen gibt es Programme um anhand von wenigen Parametern Netzwerktopologien und Datenverkehr zu generieren. Sämtliche Daten können in Form von so genannten Tracefiles aufgezeichnet und später leicht für die Auswertung genutzt werden. Insbesondere das Programm "nam" (network animator) [4] erleichtert das Verständnis der Abläufe, da es grafisch die Netzwerktopologie und den Datenverkehr darstellt.

Listing 1 [1] zeigt ein vollständiges Beispiel eines einfachen ns-2 Skriptes (in Anlehnung an [5]). Die jeweiligen Codezeilen sind bereits ausführlich kommentiert und sollten somit leicht verständlich sein. Die Grobgliederung ist wie folgt:

- Zeilen 1-29: Initialisierung der Variablen für leichte Veränderbarkeit von Parametern
- Zeilen 35-55: Definition für Protokolldateien und Programmende
- Zeilen 57-72: Definition der Eigenschaften der mobilen Knoten
- Zeilen 74-87: Definition der Netzwerktopologie und Initialisierung der mobilen Knoten
- Zeilen 89-100: Festlegung der Startpunkte und Bewegung der mobilen Knoten
- Zeilen 102-116: Festlegung der Art des Netzwerkverkehrs
- Zeilen 118-127: Festlegung des Aufrufs der Programmende-Prozedur und Start der Ausführung

Listing 1. Einfaches Beispiel eines ns-2 Skriptes (siehe example1.tcl)

```
1 # ======================================================================
2 # Define options
3 # ======================================================================
4 # Definitions of trace files
5 set val(tf)              tf.tr                    ;# name of tracefile
```

[1] Mit dieser Ausarbeitung werden die vollständigen Skripte sowie die Präsentation der Ergebnisse ausgeliefert. Bei Bedarf können Sie auch jederzeit beim Autor oder dem Institut erfragt werden.

```
set val(ntf)            ntf.nam                     ;# name of nam tracefile
        # Attention: exec command in finish(): tracefile has to be hard-coded
# ---------------------------------------------------------------------------
# Definition of mobile nodes
set val(chan)           Channel/WirelessChannel     ;# channel type
set val(prop)           Propagation/FreeSpace       ;# radio-propagation model
set val(netif)          Phy/WirelessPhy             ;# network interface type
set val(mac)            Mac/802_11                  ;# MAC type
set val(ifq)            Queue/DropTail/PriQueue     ;# interface queue type
set val(ll)             LL                          ;# link layer type
set val(ant)            Antenna/OmniAntenna         ;# antenna model
set val(ifqlen)         50                          ;# max packet in ifq
set val(rp)             AODV                        ;# routing protocol
# ---------------------------------------------------------------------------
#Definitions only relvant for NAM
set val(nodesize)       10          ;# display size of nodes in NAM
# ---------------------------------------------------------------------------
# General definitions
set val(nn)             2           ;# number of mobilenodes
set val(x)              50          ;# X dimension of the topography
set val(y)              50          ;# Y dimension of the topography
# ---------------------------------------------------------------------------
# Definitions of scheduling
set val(stoptime)       5.0         ;# time when schedule stops

# ===========================================================================
# Main Program
# ===========================================================================

# Create a simulator object
set ns_ [new Simulator]

# Open the trace file(s)
set tf [open $val(tf) w]
$ns_ use-newtrace
$ns_ trace-all $tf
set ntf [open $val(ntf) w]
$ns_ namtrace-all-wireless $ntf $val(x) $val(y)

# Define a 'finish' procedure
proc finish {} {
        global ns_ tf ntf
        $ns_ flush-trace
        #Close the NAM trace file
        close $tf
        close $ntf
        #Execute NAM on the trace file
        exec nam ntf.nam &
        exit 0
}

# Define how to create a mobile node
$ns_ node-config \
-adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
```

```
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-channelType $val(chan) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace ON \
-movementTrace ON

# Create topography
set topo [new Topography]
$topo load_flatgrid $val(x) $val(y)

# Create god
set god_ [create-god $val(nn)]

# Create nodes
for {set i 0} {$i < $val(nn)} {incr i} {
        set node_($i) [$ns_ node]
        $node_($i) random-motion 0
        # Set size of node (for NAM)
        $ns_ initial_node_pos $node_($i) $val(nodesize)
}

# Set initial node positions
$node_(0) set X_ [expr $val(x)-10.0]
$node_(0) set Y_ [expr $val(y)-10.0]
$node_(0) set Z_ 0
$node_(1) set X_ [expr $val(x)-40.0]
$node_(1) set Y_ [expr $val(y)-40.0]
$node_(1) set Z_ 0.0

# Now produce some simple node movements
$ns_ at 0.0 "$node_(0) setdest 25.0 25.0 10.0"
$ns_ at 0.0 "$node_(1) setdest 0.1 49.9 20.0"
$ns_ at 3.0 "$node_(0) setdest 40.0 15.0 35.0"

# Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns_ attach-agent $node_(0) $tcp
set sink [new Agent/TCPSink]
$ns_ attach-agent $node_(1) $sink
$ns_ connect $tcp $sink

# Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp

# Schedule events for the agents
$ns_ at 1.0 "$ftp start"
$ns_ at 2.0 "$ftp stop"

```

```
118 # Tell nodes when the simulation ends
119 for {set i 0} {$i < $val(nn) } {incr i} {
120     $ns_ at $val(stoptime) "$node_($i) reset";
121 }
122
123 # Call the finish procedure after simulation time
124 $ns_ at $val(stoptime) "finish"
125
126 # Run the simulation
127 $ns_ run
```

Indem aus Listing 1 die Codezeilen 89-101 in die Datei *example2_move*[2] und die Codezeilen 102-116 in die Datei *example2_traffic* ausgelagert werden, kann etwas mehr Übersichtlichkeit geschaffen werden. Beide Dateien werden wiederum in *example2.tcl* eingebunden. Dies wird durch den Befehl *source* und die Angabe der Datei ermöglicht. Dieser Befehl ist ähnlich dem *include* der Programmiersprache C++, bei dem Teile der Programmlogik ausgegliedert werden und somit eine bessere Wiederverwendbarkeit von Code ermöglicht wird. Die vollständige Codezeile sieht wie folgt aus:

```
source "./example2_move"
```

Der vollständige Ausschnitt mit der Beschreibung der veränderten Codezeilen zum Einbinden externer Dateien ist in Listing 2 zu sehen.

Listing 2. Einbinden von Logikteilen aus externen Dateien (siehe example2.tcl)

```
94 # Include node movement from external file
95 source "./example2_move"
101 # Include node traffic from external file
102 source "./example2_traffic"
```

Die Bewegung der Knoten in der Netzwerktopologie und der Netzwerkverkehr können auch einmalig generiert werden. Dies ist insbesondere dann hilfreich, wenn es sich um große Mengen von Knoten handelt, bei denen sowohl die Bewegung als auch der Netzwerkverkehr mehr oder weniger zufällig stattfinden sollen.

Um die Bewegung der Knoten generieren zu lassen, nutzt man das Programm *setdest* im Ordner *ns/indep-utils/cmu-scen-gen/setdest*. Dieses Programm erwartet die Angabe einiger Parameter wie z.B. die Anzahl der Knoten, deren minimale und maximale Bewegungsgeschwindigkeit und die Größe der Netzwerkfläche (*topography*) in der sich die Knoten bewegen sollen. Dabei ist darauf zu achten, dass die als Parameter verwendeten Angaben mit denen aus dem später zu verwendenden tcl-Skript übereinstimmen. Dies betrifft insbesondere die folgenden Angaben und Bezeichner:

- Anzahl der Knoten
- Größe der Topography (Fläche des Netzwerkes)
- Dauer der Simulation
- Bezeichner für die Simulation mit "ns_"
- Bezeichner für die Knoten mit "node_(<index ab 0>)"
- Bezeichner für das god mit "god_"

Da die Bewegung der Knoten in einem Gesamtzeitraum stattfindet, der zufällig zwischen 0 und 180 liegt, sollten die Gesamtlaufzeit des eigenen Skriptes auf einen Wert größer 180 gesetzt werden, um wirklich alle Bewegungen stattfinden zu lassen (siehe dazu auch [6] Kapitel XI).

Um den Netzwerkverkehr generieren zu lassen, nutzt man entweder die Datei *cbrgen.tcl* oder *tcpgen.tcl*, je nachdem ob man Datenverkehr der Art *constant bit rate* (cbr) oder *transmission control protocol* (tcp) nutzen möchte. Beide Dateien befinden sich in dem Ordner *ns/indep-utils/cmu-scen-gen/* der jeweiligen ns-2 Distribution. Auch diese erwarten wieder eine Reihe von Parametern wie die Anzahl der Knoten und die maximale Anzahl an Verbindungen.

Um die Ergebnisse dauerhaft zu speichern, werden die Ausgaben einfach in Dateien umgeleitet, statt sie ohne Angabe einer Umleitung "nur" auf dem Bildschirm angezeigt zu bekommen. Dieser Aufruf sieht dann wie folgt aus:

```
./setdest -v 2 -n 20 -s 1 -m 2.0 -M 5.0 -t 200 -P 1 -p 5.0 -x 500 -y 500 > ./
    move_20n_500sq
```

Die veränderten Codezeilen sind in Listing 3 und 4 zu sehen.

[2]Die meisten folgenden Beispiele bestehen zumeist aus drei Dateien: dem eigentlichen tcl-Skript namens "example<index>.tcl", einer Datei die alle Bewegungen der Knoten enthält "example<index>_move" sowie einer Datei die den Netzwerkverkehr enthält "example<index>_traffic". Gibt es mehrere solcher Dateien, wurde versucht einen eindeutigeren Dateinamen zu wählen.

Listing 3. Anpassen der Variablen für die automatische Generierung von Knotenbewegung und Netzwerkverkehr (siehe example3.tcl)

```
# General definitions
set val(nn)             20          ;# number of mobilenodes (-n)
set val(x)              500         ;# X dimension of the topography (-x)
set val(y)              500         ;# Y dimension of the topography (-y)
# ---------------------------------------------------------------------------
# Definitions of scheduling
set val(stoptime)       200.0       ;# time when schedule stops (-t)
```

Listing 4. Einbinden der generierten Dateien mit Erklärungen zur Nutzung (siehe example3.tcl)

```
# ======================================================================
# Generate random movement by calling 'setdest' in
# ~ns/indep-utils/cmu-scen-gen/setdest with several parameters
# and pipe output to external (text) file
# E.g. use
# ./setdest -v 2 -n 20 -s 1 -m 2.0 -M 5.0 -t 200 -P 1 -p 5.0 -x 500 -y 500 > ./
    move_20n_500sq
# and then copy file to local source folder
# Nearly all used parameters will be shown in the output file, check for validity!
# Parameters:
#       -v <version> [use '2' to specify more parameters]
#       -n <number of nodes> [set equal to number of nodes in this script!]
#       -s <speed type> [unknown, just use '1']
#       -m <minimum speed>
#       -M <maximum speed>
#       -t <simulation time> [set equal to simulation time in this script!]
#       -P <pause type> [unknown, just use '1']
#       -p <pause time>
#       -x <maximum X> [set equal to X-size of topography in this script!]
#       -y <maximum Y> [set equal to Y-size of topography in this script!]
#
# More help: http://www.isi.edu/nsnam/ns/tutorial/nsscript7.html#second
#
# Include node movement from external file
source "./move_20n_500sq"
#source "./move_100n_500sq"

# Generate random traffic by calling 'cbrgen.tcl' or 'tcpgen.tcl' in
# ~ns/indep-utils/cmu-scen-gen/ with several parameters
# and pipe output to external (text) file
# E.g. use
# ns cbrgen.tcl -type cbr -nn 20 -seed 1.0 -mc 15 -rate 10.0 > ./traffic_20n_cbr
# and then copy file to local source folder
# Nearly all used parameters will be shown in the output file, check for validity!
# Parameters:
#       -type <cbr | tcp>
#       -nn <number of nodes> [set equal to number of nodes in this script!]
#       -seed <seed>
#       -mc <maximum number of connections>
#       -rate <rate>
#
# More help: http://www.isi.edu/nsnam/ns/tutorial/nsscript7.html#first
#
# Include node traffic from external file
source "./traffic_20n_cbr"
#source "./traffic_100n_cbr"
```

```
141
142 # For both generating methods take care of naming your own variables
143 # equal to that generated by the scripts:
144 # - 'ns_' for the simulator
145 # - 'node_(<index>)' for the nodes
146 # - 'god_' for the god
147 # - (and maybe others ...)
148 # ================================================================
```

Listing 5. Auszüge der generierten Knotenbewegungen (siehe move_20n_500sq)

```
1 #
2 # nodes: 20, speed type: 1, min speed: 2.00, max speed: 5.00
3 # avg speed: 3.08, pause type: 1, pause: 5.00, max x: 500.00, max y: 500.00
4 #
5 $node_(0) set X_ 381.805793246162
6 $node_(0) set Y_ 294.899763269152
7 $node_(0) set Z_ 0.000000000000
8 $node_(1) set X_ 191.320651628705
64 $node_(19) set Z_ 0.000000000000
65 $ns_ at 0.000000000000 "$node_(0) setdest 436.603932947297 470.358797194117
       3.391692426864"
66 $ns_ at 0.000000000000 "$node_(1) setdest 98.815086193741 64.138980312484
       3.069414356593"
83 $ns_ at 0.000000000000 "$node_(19) setdest 217.888581990456 218.060241280247
       4.639801815171"
84 $god_ set-dist 0 1 1
85 $god_ set-dist 0 2 1
273 $god_ set-dist 18 19 2
274 $ns_ at 1.392757125219 "$god_ set-dist 6 10 1"
275 $ns_ at 1.392757125219 "$god_ set-dist 6 13 2"
324 $ns_ at 18.802511310027 "$god_ set-dist 13 14 2"
325 $ns_ at 20.164896887747 "$node_(16) setdest 166.179963169953 383.977305462416
       0.000000000000"
326 $ns_ at 20.700818136627 "$god_ set-dist 7 19 1"
770 $ns_ at 199.236567050032 "$god_ set-dist 15 19 2"
771 #
772 # Destination Unreachables: 0
773 #
774 # Route Changes: 415
775 #
776 # Link Changes: 340
777 #
778 # Node | Route Changes | Link Changes
779 #    0 |            53 |           36
780 #    1 |            50 |           44
781 #    2 |            33 |           33
782 #    3 |            39 |           34
783 #    4 |            51 |           35
784 #    5 |            43 |           40
785 #    6 |            45 |           36
786 #    7 |            42 |           34
787 #    8 |            43 |           30
788 #    9 |            36 |           36
789 #   10 |            51 |           43
790 #   11 |            33 |           30
791 #   12 |            31 |           31
```

```
#    13 |              60 |              23
#    14 |              38 |              32
#    15 |              39 |              32
#    16 |              29 |              28
#    17 |              38 |              37
#    18 |              36 |              30
#    19 |              40 |              36
#
```

Listing 6. Auszüge des generierten Netzwerkverkehrs (siehe traffic_20n_cbr)

```
#
# nodes: 20, max conn: 15, send rate: 0.10000000000000001, seed: 1.0
#
#
# 1 connecting to 2 at time 2.5568388786897245
#
set udp_(0) [new Agent/UDP]
$ns_ attach-agent $node_(1) $udp_(0)
set null_(0) [new Agent/Null]
$ns_ attach-agent $node_(2) $null_(0)
set cbr_(0) [new Application/Traffic/CBR]
$cbr_(0) set packetSize_ 512
$cbr_(0) set interval_ 0.10000000000000001
$cbr_(0) set random_ 1
$cbr_(0) set maxpkts_ 10000
$cbr_(0) attach-agent $udp_(0)
$ns_ connect $udp_(0) $null_(0)
$ns_ at 2.5568388786897245 "$cbr_(0) start"
#
# 15 connecting to 17 at time 43.420613009212822
#
set udp_(14) [new Agent/UDP]
$ns_ attach-agent $node_(15) $udp_(14)
set null_(14) [new Agent/Null]
$ns_ attach-agent $node_(17) $null_(14)
set cbr_(14) [new Application/Traffic/CBR]
$cbr_(14) set packetSize_ 512
$cbr_(14) set interval_ 0.10000000000000001
$cbr_(14) set random_ 1
$cbr_(14) set maxpkts_ 10000
$cbr_(14) attach-agent $udp_(14)
$ns_ connect $udp_(14) $null_(14)
$ns_ at 43.420613009212822 "$cbr_(14) start"
#
#Total sources/connections: 10/15
#
```

V. Implementierung von Bösartigkeit

Um nun die vorher beschrieben bösartigen Verhaltensweisen zu implementieren, wurden verschiedene Verfahren ausprobiert. Die Ergebnisse werden auszugsweise vorgestellt.

A. DelayBox

Die DelayBox [7] ist ein bereits in ns-2 implementierter Knotentyp, der zwischen einem Sender und einem Empfänger platziert wird. Der Datenfluss wird dabei vom Sender zur DelayBox und von der DelayBox zum Empfänger eingerichtet. Das Verhalten der DelayBox wird dabei konkret auf einen Datenfluss (*flow*) angewandt und nicht auf Basis von Paketen. Dies

bedeutet, dass jeder Datenfluss zwischen Sender und Empfänger einen eindeutigen Bezeichner besitzt, durch dessen Überprüfung die DelayBox entscheidet, ob sie den Datenfluss stört oder nicht. Dabei kann die DelayBox nur auf TCP Datenverkehr angewandt werden. Das Verhalten der DelayBox wird durch zwei Tabellen bestimmt. Die erste Tabelle ist die so genannte *rule table* und enthält die durch den Benutzer erstellten Werte, die das Verhalten der DelayBox beeinflussen. Die zweite Tabelle ist die *flow table*, in der interne Einstellungen zur Anwendung der Regeln auf den Datenstrom eingetragen sind.

Bei der Implementierung einer DelayBox werden typischerweise drei Variablen definiert: die Verzögerung beim Weiterleiten von Paketen (*delay*), die Bandbreite der DelayBox und die Verlustrate (*loss*). Anschließend wird die DelayBox ebenso wie ein anderer Knoten erstellt und eine Regel definiert, durch die die Variablen als Attribute der DelayBox bestimmt werden. Dies ist in Listing 7 dargestellt.

Listing 7. Definition von Eigenschaften der DelayBox (siehe example4.tcl)

```
81 # Create random variables for the behaviour of the delaybox
82 set delay [new RandomVariable/Uniform]        ;# delay 2-4 ms
83 $delay set min_ 2
84 $delay set max_ 4
85 set bw [new RandomVariable/Constant]          ;# bw 100 Mbps
86 $bw set val_ 100
87 set loss_rate [new RandomVariable/Uniform]    ;# loss 50-75% loss
88 $loss_rate set min_ 0.50
89 $loss_rate set max_ 0.75
99 # Create DelayBox nodes
100 set db_(0) [$ns_ DelayBox]
111 # Setup rules for DelayBoxes
112 $db_(0) add-rule [$node_(0) id] [$node_(1) id] $delay $loss_rate $bw
```

Außerdem ist es notwendig bei der Einrichtung des Netzwerkverkehrs die DelayBox so zu platzieren, wie in Listing 8 dargestellt.

Listing 8. Platzieren der DelayBox (siehe example4_traffic)

```
2 # Setup Links between tcp-source, delayBox and tcp-sink
3 $ns_ duplex-link $node_(0) $db_(0) 100Mb 1ms DropTail
4 $ns_ duplex-link $db_(0) $node_(1) 100Mb 1ms DropTail
```

Die DelayBox unterliegt jedoch einigen Beschränkungen, die ihren Einsatz für die angestrebte Implementierung der Feedbackmechanismen schwer möglich machen:

- für jedes Sender-Empfänger-Paar muss eine DelayBox eingerichtet werden
- der Einsatz der DelayBox beschränkt sich auf den Datenfluss und kann nicht auf Paket-Basis eingerichtet werden
- durch das feste Setzen der Verbindung werden die Entfernungen der Knoten zweitrangig, was nicht der Realität von mobilen Knoten mit beschränkten Sendereichweiten entspricht

Details dazu können in der Dokumentation [8] Kapitel 22 nachgelesen werden.

B. ErrorModel

Das ErrorModel ist ebenfalls bereits in ns-2 implementiert. Es stellt keinen eigen Knotentyp dar, sondern simuliert Fehler auf der Verbindung (*link*) zwischen zwei Knoten. Dabei können einfache Fehlermodelle wie Fehlerraten von Paketen ebenso berücksichtigt werden wie statistische oder empirische Fehlermodelle.

Das einfache ErrorModel kann mit verschiedenen Einheiten wie Paket, Zeit oder Bit arbeiten. Die Anwendung auf Paketebene ist wohl die sinnvollste und leider auch die einzig ansatzweise dokumentierte. Dabei wird je nach Angabe der Häufigkeit (*rate*) das *Error-Flag* des Paketes gesetzt, um es als fehlerhaft zu markieren. Die Folge ist, dass es beim Empfänger als fehlerhaft erkannt und somit nicht angenommen wird. Dies entspricht dem Verlust des Paketes. Die dritte Angabe zum ErrorModel (neben der Verlustrate und der Einheit) ist die Angabe eines Ziels, an die das fehlerhafte Paket weitergeleitet werden soll. Um es einfach "'verschwinden"' zu lassen, reicht die Angabe eines so genannten Null-Agenten aus. Das Anwenden dieses ErrorModel kann weiterhin durch die optionale Angabe einer Zufallsvariable beeinflusst werden, die bei Nicht-Angabe automatisch auf einen Wert zwischen 0 und 1 genormt wird.

In der Datei mit den Daten des Netzwerkverkehrs wird, wie in Listing 9 dargestellt, die Definition des ErrorModel eingefügt.

Listing 9. Definieren eines ErrorModel (siehe example5_traffic)

```
6 set error_module_variable [new RandomVariable/Uniform]
7 $error_module_variable set min_ 0.0
```

```
$error_module_variable set max_ 1.0

set error_module [new ErrorModel]                ;# create the error model
$error_module set rate_ 0.02                     ;# set error rate
$error_module unit pkt                           ;# error unit: packets
$error_module ranvar $error_module_variable      ;# attach random variable to error
    module
$error_module drop-target [new Agent/Null]       ;# set target for dropped packets

$ns_ lossmodel $error_module $node_(0) $node_(1)     ;# set error model between nodes
```

Die Anwendung eines ErrorModel auf Pakete kann vor oder nach der Einreihung in die knoteninterne Paketwarteschlange erfolgen. Im Beispiel in Listing 9 wird das ErrorModel vor der Warteschlange angewandt. Durch Anwendung von *link-lossmodel* statt *lossmodel* kann das ErrorModel auch nach der Warteschlange angewandt werden. Allerdings ist auch die Anwendung des ErrorModel noch nicht zufriedenstellend, da ähnlich der DelayBox sehr viel manuelle Arbeit zur Einrichtung aller gewünschten Abweichungen notwendig ist und der Nutzen immer noch auf einen geringen Umfang beschränkt ist. Details dazu können in der Dokumentation [8] Kapitel 13 nachgelesen werden.

C. Routing-Protokoll

Eine optimale Einrichtung der Abweichungen ist durch eine eigene Implementierung der Bösartigkeit besser zu erreichen. Dazu werden Veränderungen am Routing-Algorithmus vorgenommen. Dies ist zwar deutlich aufwendiger, da neben einem grundlegenden Verständnis für die Routing-Algorithmen vor allem die zumeist kaum dokumentierten ns2-Klassen (Ausnahme in [9]) und die Implementierung der Routing-Algorithmen nachvollzogen werden müssen. Es ist nach dieser Einarbeitung jedoch auch deutlich einfacher, zielgerichtete Veränderungen vorzunehmen, die zur Implementierung der Feedbackmechanismen aus [3] notwendig sind. Im Folgenden soll die grundlegende Idee dieser Umsetzung beschrieben werden. Details sind auf der Website der Lehrveranstaltung [10] in anderen Ausarbeitungen zu finden.

Um festzulegen, welche Knoten welchen Grad an Bösartigkeit haben sollen, muss zu Beginn des Programms eine Definition bereitgestellt werden. Dafür wird zum einen C++-Code benötigt, der diese Werte aus einer Textdatei zu Programmstart einliest und zum anderen eben diese Textdatei, die enthält, welcher Knoten zu welchem Grad bösartig ist. Diese Werte werden in einem globalen Speicher vorgehalten, auf den alle Knoten jederzeit zugreifen können. Alternativ kann auch jeder Knoten selbst die Daten vorhalten. Dies richtet sich in erster Linie nach der Implementierung des jeweils verwendeten Routing-Algorithmus. Auf die Darstellung dieser beiden Dateien wurde hier aus Platzgründen verzichtet.

Jedes empfangene Paket wird bei jedem Knoten durch eine Prozedur überprüft. Dabei wird z.B. der Empfänger aus den Kopfdaten des Pakets ausgelesen und anhand der interne Routingtabelle wird das Paket dorthin weitergeleitet. Vor diesem Zugriff muss der Knoten prüfen, ob er bösartig ist oder nicht. Falls er bösartig ist, entscheidet sich durch den Grad seiner Bösartigkeit wie er dieses Paket behandelt. Dabei sind prinzipiell nur zwei Auswahlmöglichkeiten angedacht; die Entscheidung

- ob das Paket weitergeleitet wird oder nicht => das Löschen wird somit simuliert und
- wann das Paket weitergeleitet wird => das Verzögern wird simuliert.

Bei der Verzögerung gilt es zusätzlich einen Timer zu implementieren, der aus einer festzulegenden Zeitdifferenz und der aktuellen Zeit den Zeitpunkt für die Weiterleitung bestimmt und diese dann auch vornimmt. Weitere Details zu der Implementierung lassen sich in den anderen Ausarbeitungen unter [10] finden. Interessante Hinweise zum Verändern von ns2-Implementierungen finden sich u.a. in [11], [12], [13], [14] und [15]. Das Nutzen der bereits vorhandenen Implementierung der Blacklist scheint ein weiterer interessanter Ansatz zu sein um Bösartigkeit zu implementieren (siehe dazu u.a. [16]).

VI. Sendereichweite mobiler Knoten

Ein weiteres Problem stellt die Definition der Sendereichweite mobiler Knoten dar. Mobilen Knoten kann eine Empfangsstärke mittels *set RXThresh_ <value>* attributiert werden. Im Order *ns/indep-utils/propagation/threshold* befindet sich das Programm *threshold*, das nach dem Kompilieren unter Angabe des Übertragungsmodells (*propagation-model*) und der gewünschten Entfernung in Metern (*distance*) als Parameter die notwendigen Werte berechnet. Der minimale Befehl lautet wie folgt:

```
threshold -m <propagation-model> distance
```

Diese berechneten und angezeigten Werte können dann bei der Knotendefinition im tcl-Skript eingetragen werden. Problematisch erscheint dabei, dass das Übertragungsmodell *FreeSpace* offensichtlich alle Einstellungen dazu zu ignorieren scheint. Selbst Minimalbeispiele konnten die Funktionalität nicht prüfen. Erst nach einer Umstellung auf das eigentlich komplexere Übertragungsmodell *TwoRayGround* konnte die Funktionalität bestätigt und genutzt werden. Allerdings schienen auch nicht alle durch das Programm *threshold* bestimmten Entfernungen einwandfrei zu funktionieren. Einstellungen für eine Reichweite von

100m blieben ohne Einfluss auf das weiterhin stattfindende Senden und Empfangen von Knoten mit einer Entfernung größer als 100m. Bei Nutzung einer Sendereichweite für 250m arbeitete jedoch alles wie vorgesehen: Knoten mit einer Entfernung von 251m konnten danach keine Pakete mehr zueinander senden. Hinweise zu Einstellung der Sendereichweite mobiler Knoten finden sich u.a. in [8] im Kapitel 18.4, [17], [18] und [19].

Sowohl für die Einrichtung der DelayBox als auch des ErrorModel sei noch erwähnt, dass das automatische Einrichten von Links wie z.B.

```
$ns_ duplex-link $node_(0) $node_(1) 100Mb 1ms DropTail
```

durch die Programme *cbrgen.tcl* und *tcpgen.tcl* zu festen Verbindungen von Knoten führt, die jegliche Angaben von Entfernungen und Sendereichweiten ignorieren. Zur Lösung reicht es, in den entsprechenden Dateien die Zeilen mit dem Aufbau der Verbindungen zwischen den einzelnen Knoten auszukommentieren (mittels "#") oder zu löschen. Da die DelayBox jedoch eben auf Link-Ebene arbeitet, scheint ein gleichzeitiges Nutzen der DelayBox sowie das Unterbrechen von Verbindungen aufgrund sich ändernder Entfernungen der mobilen Knoten unmöglich zu sein. Allgemeine Hinweise zu vielen ns Problemen lassen sich über das Forum [20] oder die Suche in den ns Emaillisten [21] finden.

VII. Zusammenfassung

Bösartigkeit hat zahlreiche Ursachen, die meist sehr komplex und nur schwer einzugrenzen sind. Die Ergebnisse der Bösartigkeit lassen sich jedoch in einer einfachen Form wie dem Verlust von Paketen gut messen. Dazu müssen einfache Wege gefunden werden, den Verlust von Paketen in Netzen verschiedener Größe bei nur geringem Änderungsaufwand zu simulieren. Einige Möglichkeiten wie der Einsatz der DelayBox, des ErrorModel und auch die Idee einer Implementierung im Routing-Protokoll wurden vorgestellt. Die bisherigen Ergebnisse lassen sich noch weiter verfeinern, um die Implementierung der Feedbackmechanismen aus [3] vorzunehmen. Wichtig zu erwähnen ist, dass bei Nutzung des Routing-Protokolls AODV keine vollständig, d.h. RFC3561 [22] konforme, Implementierung in ns-2 vorliegt.

Ein weiterer Punkt für Überlegungen zur Implementierung von Bösartigkeit ist, welche Pakete von dieser Bösartigkeit betroffen sein sollen. Sind es nur Datenpakete oder auch Routingpakete? Gerade der Verlust von Routingpaketen kann zu unvorhersehbaren Verzögerungen oder gar Verlusten beim Versand von Datenpaketen führen. Diese Möglichkeit kann jedoch auch zur Überprüfung der Wirksamkeit von Feedbackmechnismen sinnvoll eingesetzt werden, wenn sie leicht kontrollierbar ist. Das Spektrum der weiteren Möglichkeiten die Wirksamkeit von Protokollen und Mechanismen zu überprüfen, indem gezielt Abweichungen implementiert werden, eröffnet ein weites Feld für Veränderungen an der Software-Implementierung zur Simulation solcher Netzwerke.

Der Einsatz von DelayBox und ErrorModel für sehr kleine Netze mit geringen bis keinen Änderungen ist praktikabel. Allerdings sind beide Möglichkeiten auch wieder auf die Nutzung von nicht mobilen Knoten beschränkt, da sie auf Verbindungen zwischen Knoten basieren. Die Implementierung durch Veränderungen an einem Routing-Protokoll bildet eine bessere Basis für individuelle Umsetzungen der Bösartigkeit. Jedoch ist auch hier zu beachten, dass bei Nutzung eines anderen Routing-Protokolls der gleiche Mehraufwand an erstmaliger Implementierung entsteht, wenn keine Möglichkeit gefunden wird, die Implementierung von Bösartigkeit protokollunabhängig vorzunehmen.

References

[1] "Internet rechtslexikon 'lexexakt.de'." [Online]. Available: http://www.lexexakt.de/glossar/dolus2.php
[2] "The Network Simulator ns-2." [Online]. Available: http://www.isi.edu/nsnam/ns
[3] B. Schuenemann, C. Meinel, and S. Linckels, "Feedback-based solution for avoiding attacks on mobile ad-hoc networks," 2006.
[4] "Nam - Network Animator." [Online]. Available: http://www.isi.edu/nsnam/nam
[5] "NS by Example." [Online]. Available: http://nile.wpi.edu/NS/
[6] "Tutorial for the Network Simulator ns." [Online]. Available: http://www.isi.edu/nsnam/ns/tutorial/index.html
[7] M. C. Weigle. (2005) Per-flow delay and loss. Old Dominion University, Dept. of Computer Science. [Online]. Available: http://dirt.cs.unc.edu/delaybox/
[8] "The Network Simulator ns-2 Documentation (ns manual)." [Online]. Available: http://www.isi.edu/nsnam/ns/ns-documentation.html
[9] "ns2 Network Simulator (ITM-cvs) Documentation." [Online]. Available: http://www.auto-nomos.de/ns2doku/main.html
[10] "Website der Lehrveranstaltung." [Online]. Available: http://www.hpi.uni-potsdam.de/~meinel/teaching/attacken_und_sicherheitsstrategien_in_mobilen_mesh-_und_ad-hoc-netzen_ws200607.html
[11] F. J. Ros and P. M. Ruiz, "Implementing a New Manet Unicast Routing Protocol in NS2," http://masimum.dif.um.es/nsrt-howto/pdf/nsrt-howto.pdf, May 2007.
[12] "ns-2 debugging help." [Online]. Available: http://www.cs.ust.hk/~cszyz/ns2-debug.html
[13] "Pedro Vale Estrela - NS2 page." [Online]. Available: http://tagus.inesc-id.pt/~pestrela/ns2/
[14] "Bryan's NS-2 DSR FAQ." [Online]. Available: http://www.geocities.com/b_j_hogan
[15] "ns2 FAQ with collected answers from ns mailing list." [Online]. Available: http://web.syr.edu/~dchen02/FAQ.txt
[16] "Crossroads ACM Student Magazine about blacklist implementation." [Online]. Available: http://www.acm.org/crossroads/xrds11-1/adhoc.html
[17] "How to set the communication radius in wireless nodes?" [Online]. Available: http://140.116.72.80/~smallko/ns2/range_en.htm
[18] "Regarding AODV and range (from ns mailing list)." [Online]. Available: http://mailman.isi.edu/pipermail/ns-users/2004-July/043584.html
[19] "ns2 FAQ from Arijit Ghosh." [Online]. Available: http://www.ics.uci.edu/~arijitg/faq.html
[20] "Nabble Forum Network Simulator ns-2." [Online]. Available: http://www.nabble.com/Network-Simulator-ns-2-f15582.html
[21] "NsNam Site Search." [Online]. Available: http://www.isi.edu/nsnam/htdig/search.html
[22] "RFC 3561: Ad hoc On-demand Distance Vector (AODV) Routing." [Online]. Available: http://ietf.org/rfc/rfc3561.txt