

<i>Routing (Visualisierung)</i>	als Beispiel vorhanden, kann leicht angepasst werden
<i>Shortest Path Algorithmus</i>	gibts noch nicht
<i>Ausfallen einzelner Knoten</i>	selbst zu implementieren
<i>Kollisionen</i>	selbst zu implementieren
<i>ALOHA</i>	pure Aloha (overload) pure Aloha (optimal load) pure Aloha (low traffic) slotted Aloha (overload) slotted Aloha (optimal load) slotted Aloha (low traffic)
<i>Hardware</i>	selbst zu implementieren
<i>Übertragungsfehler (CRC, Verlust)</i>	selbst zu implementieren
<i>Installation</i>	Linux – mittel, Windows – sehr leicht dafür schafft man das Integrieren eigener Module im Windows nicht
<i>User Interface</i>	Sehr gut
<i>API</i>	Sehr gut dokumentiert Timer, Nachrichtenübermittlung, Outputs, GUI

Vorteile:

- einfacher Modularer Aufbau mit nur 2 Methoden pro Modul
- nur 2 verschiedene Bausteine – Connections, Modules
- sehr viele Anwendungsmöglichkeiten
- Grafischer Editor für die Moduldefinition
- gute Outputmöglichkeiten (Sprechblasen, Eventlogger, Outputfiles, Farben)

Nachteil:

- alles bis auf Simulationsgrundfunktionen muss selbst definiert und implementiert werden

OPNet Academic Guru [Windows, gar nicht zu erweitern]

<i>Routing (Visualisierung)</i>	gibts schon
<i>Shortest Path Algorithmus</i>	OSPF in verschiedenen Varianten
<i>Ausfallen einzelner Knoten</i>	Knoten können während der Simulation zum Fail gezwungen werden
<i>Kollisionen</i>	
<i>ALOHA</i>	
<i>Hardware</i>	Sehr sehr viel nach Einsatzgebiet und Hersteller sortiert vorhanden (jeweils mit Kurzbeschreibung)
<i>Übertragungsfehler (CRC, Verlust)</i>	
<i>Installation</i>	Sehr leicht
<i>User Interface</i>	Sehr gut
<i>API</i>	keine gefunden

Nachteile:

- kaum Dokumentation
- nur 1 einziges Tutorial – Beispiel
- keine Hinweise auf Erweiterbarkeit, außer selbst zusammenstellbaren Szenarien

Vorteile:

- sehr viel vordefinierte Szenarien
- viele Möglichkeiten zum Zusammenstellen eigener Szenarien (ohne Programmieren)
- über das User-Interface gut nachvollziehbar
- sehr viele Statistiken

<i>Routing (Visualisierung)</i>	gibts schon
<i>Shortest Path Algorithmus</i>	selbst zu implementieren
<i>Ausfallen einzelner Knoten</i>	Knoten können während der Laufzeit zum Reboot, Crash, Shutdown, Pause, Fail gezwungen werden
<i>Kollisionen</i>	selbst zu implementieren
<i>ALOHA</i>	gibt noch nicht
<i>Hardware</i>	alles wird nur auf Hosts und Router abgebildet
<i>Übertragungsfehler (CRC, Verlust)</i>	durch Framefarbe dargestellt
<i>Installation</i>	Einfach
<i>User Interface</i>	Sehr gut

<i>Routing (Visualisierung)</i>	gibts schon
<i>API</i>	<p>kurz dokumentiert</p> <ul style="list-style-type: none"> ● Application Layer functions. <ul style="list-style-type: none"> ● CNET read application ● CNET write application ● CNET enable application ● CNET disable application ● Physical Layer functions. <ul style="list-style-type: none"> ● CNET write physical ● CNET write physical reliable ● CNET write direct ● CNET read physical ● CNET set promiscuous ● CNET set nicaddr ● CNET parse nicaddr ● CNET format nicaddr ● Timer functions. <ul style="list-style-type: none"> ● CNET start timer ● CNET stop timer ● CNET timer data ● Tracing functions <ul style="list-style-type: none"> ● CNET trace ● CNET trace name ● CNET set trace ● CNET get trace ● Event handling functions. <ul style="list-style-type: none"> ● CNET set handler ● CNET get handler ● Checksum and miscellaneous functions. <ul style="list-style-type: none"> ● checksum internet ● checksum ccitt ● checksum crc16 ● checksum crc32 ● CNET read keyboard ● CNET set debug string ● CNET set time of day

Vorteile:

- Schichtaufbau
- Frame – Visualisierung

Nachteile:

- eigene Topologiedefinitionssprache, kaum Dokumentation dazu
- nur Physical Layer und Application Layer schon vorhanden (+ Error Layer)
- sehr wenig Dokumentation

ns-2 [Linux, windows-cygwin, C++, tcl-scripts]

Routing (Visualisierung)	als Beispiel vorhanden, kann leicht angepasst werden
Routing Algorithmen	Static, Distance Vector und Link-State
Ausfallen einzelner Knoten	Per Event steuerbar
Kollisionen	GUI-Editor bietet Keine Steuerung auf physikalischer Ebene
ALOHA	Nicht vorhanden, Der GUI-Editor erlaubt nur Schichten ab IP- Ebene
Hardware	Alles über "universal"-nodes, zu denen ein Protokollstack angehängt wird (UDP, TCP, CBR, Telnet, HTTP, FTP...)
Übertragungsfehler (CRC, Verlust)	Nicht möglich (GUI-Editor Restriktionen), aber die Queue-Kapazität jedes "universal"-nodes ist einstellbar (Anzahl der Pakete)
Installation	Linux – mittel, da es ein umfangreiches Package ist; Windows – nicht probiert,
User Interface	Bietet nicht die volle Funktionalität der tcl- scripts
API	gut dokumentiert, aber sehr umfangreich ein eigenes Protokoll zu schreiben ist nicht schwer (Kommando-Interpreter "command()"), Message-Handler "recv()", Initialisierung mit Konstruktor), aber es zu integrieren schon

Vorteile:

- sehr viele Anwendungsmöglichkeiten
- weite Verbreitung, viele Tutorials
- Animation von ausgeführten Simulationen voll eingebunden

Nachteil:

- sehr umfangreich
- Grafischer Editor kann die Möglichkeiten des Simulators nicht ausschöpfen
- Beim Erweitern des Simulators zB um ein eigenes Routing-Protokoll muss sowohl in C++, als auch in Tcl entwickelt werden.

NCtuns [Linux (RPM für Fedora), C++]

Routing (Visualisierung)	als Beispiel vorhanden, kann leicht angepasst werden kleines Minus: Alle Pakete heißen "DATA"
Routing Algorithmen	Static, RIP und OSPF, Fehlverhalten bei Simulation
Ausfallen einzelner Knoten	Per Event steuerbar
Ausfallen einzelner Links	Per Event steuerbar
Kollisionen	Auf MAC-Ebene über Log-file einsehbar, GUI-Visualisierung??
ALOHA	Nicht vorhanden (aber Mobile Geräte mit zB Ad hoc routing vorhanden)
Hardware	Hub, Switch, Router, Host jede Komponente hat eigenen Protokollstack
Übertragungsfehler (CRC, Verlust)	Bit-Error-Rate für Links einstellbar, Queue-Kapazität jeder Hardware-Komponente ist einstellbar (Anzahl der Pakete)
Installation	Erfordert leider Kernel-Patch, GUI-Komponente und Simulator-Komponente sind getrennt
User Interface	Sehr gut, voll integriert
API	+ Module (ARP, Package-Queueing, MAC-Ebene) sind gut dokumentiert und Erweiterbarkeit auch dokumentiert - Tools (RIP, OSPF) schlecht dokumentiert, nur Source-Code vorhanden

Vorteile:

- sehr viele Anwendungsmöglichkeiten
- viele Beispiele
- Animation von ausgeführten Simulationen voll eingebunden
- Topologie-Erstellung läuft generell über GUI-Editor
- Linux-eigener TCP/IP – Protokollstack wird bei Simulation ausgeführt

Nachteil:

- umfangreich
- schwierige Installation
- Routing Algorithmen nicht als Module eingebunden

Real5.0 [Linux, Unix..., Simulator: C, GUI: Java, Netlanguage-Scripts]

Routing (Visualisierung)	als Beispiel vorhanden, keine Visualisierung, nur Dump und Plot files
Routing Algorithmen	Nicht vorhanden
Ausfallen einzelner Knoten	?
Kollisionen	CSMA-Modul vorhanden (Ethernet)
ALOHA	Nicht vorhanden
Hardware	Unterscheidung zwischen Nodes und Routern
Übertragungsfehler (CRC, Verlust)	?
Installation	Linux – mittel, da es für Unix gedacht ist, aber ein kleines Package ist
User Interface	Einfach gehalten, wenige Parameter, keine Animationskomponente
API	dokumentiert

Vorteile:

- einfach (ca. 30 Module)
- Erweitern dürfte leicht fallen

Nachteil:

- keine Animation der Simulationen, nur Dump und Plot-Files, um sich den Verlauf (Dump) der Simulation und die Variablen(Plot) anzuschauen
- Grafischer Editor etwas mickrig

SSFNet [Linux, windows, C++/Java, ned-scripts]

Routing (Visualisierung)	BGP(Boarder Gateway Protokoll)-Animation verfügbar
Routing Algorithmen	OSPF, BGP
Ausfallen einzelner Knoten	?
Kollisionen	?
ALOHA	Nicht vorhanden
Hardware	Hosts, Router...
Übertragungsfehler (CRC, Verlust)	?
Installation	Linux – leicht, zumindest das Java-Package; Windows – nicht probiert,
User Interface	Animator/Topology-Viewer schlecht dokumentiert, Animation zeigt keinen Paketfluss GUI-Editor (GLASS) zu wenige Parameter, um nicht-optische Simulation zu konfigurieren
API	gut dokumentiert, Erweiterbarkeit leicht möglich

Vorteile:

- leichte Erweiterbarkeit

Nachteil:

- umfangreich, schlechte Dokumentation
- kein oder unzureichender GUI-Editor (GLASS)
- Animationskomponente muss für jede Simulation ausprogrammiert werden (Player.java)
- C++ Variante hat nicht so viele Module