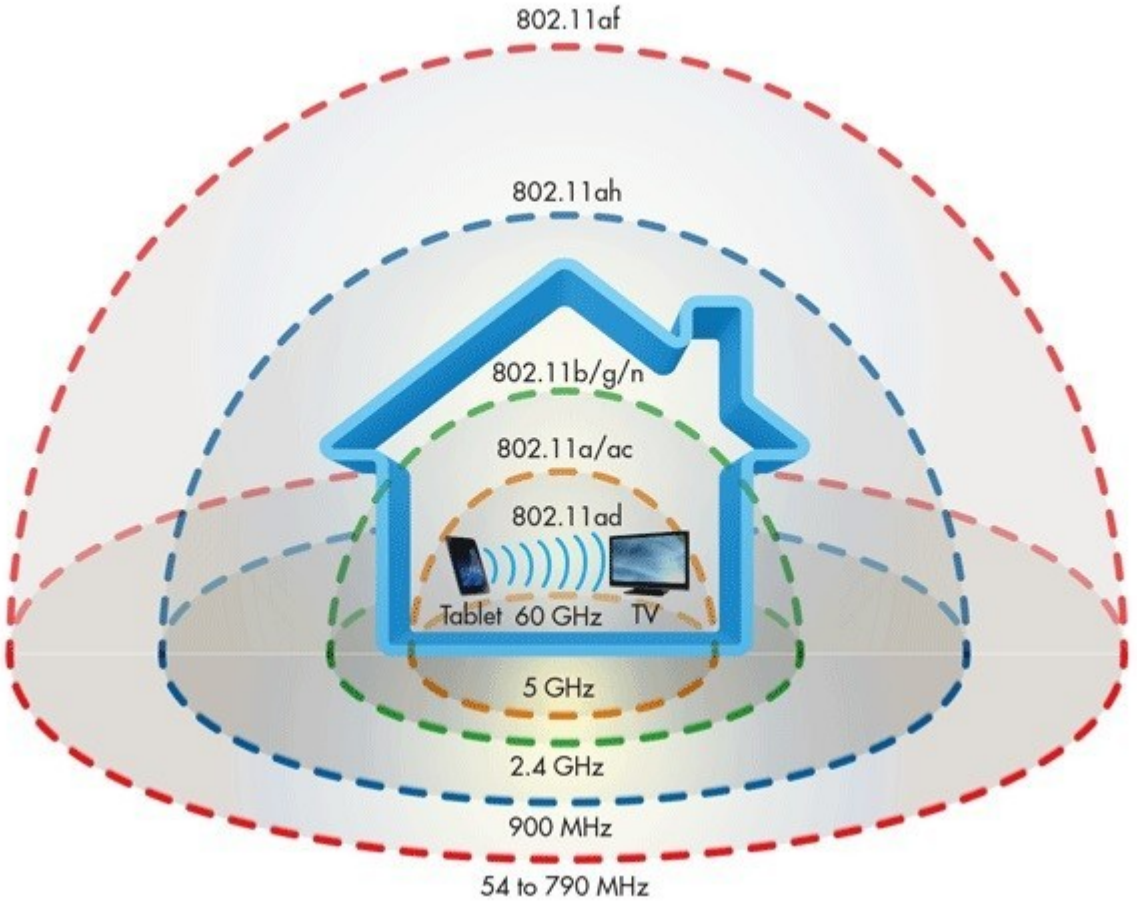
STU FEI

Počítačové siete 2

Modely pre generovanie paketov, simulačné nástroje

Martin Drozda

802.11



V zmysle modelov šírenia signálu, napr. free-space modelu, platí, že čím vyššia frekvencia, tým väčší je opad signálu so vzdialenosťou. Pri 60GHz je opad najväčší.

Modely pre generovanie paketov

Modely pre generovanie dátových paketov sú potrebné pre modelovanie vzniku týchto paketov. V reálnej sieti vznikajú napr. používaním prehliadača alebo emailového klienta.

V simulácii je potrebné vznik dátových paketov modelovať pomocou vhodného štatistického rozdelenia.

Často používaný je Poissonov proces.

Senzory môžu generovať dátové pakety v konštatných intervaloch, vtedy je možné použiť CBR (Constant bit rate).

VBR = variable bit rate
Generovanie paketov podľa daného štatistického rozdelenia (ale nie uniformne)

Modely pre generovanie paketov

CBR = constant bit rate
Generovanie paketov v konštantných intervaloch

Napr. Glomosim:

CBR 947 1174 4000 128 0.025S 0S 100S

V tomto prípade je nový dátový paket generovaný každú 0.025 sekundu, teda 40 paketov za sekundu.

Modely pre generovanie paketov

Príklady pre ns3:

```
OnOffHelper onOffHelper ("ns3::UdpSocketFactory",
InetSocketAddress (Ipv4Address ("10.0.0.2"), cbrPort));

onOffHelper.SetAttribute ("OnTime",  StringValue
("ns3::ConstantRandomVariable[Constant=1]"));

onOffHelper.SetAttribute ("OffTime", StringValue
("ns3::ConstantRandomVariable[Constant=0]"));
```

Pakety sú generované počas „on“ stavu.

Modely pre generovanie paketov

Príklady pre ns3:

```
onOffHelper.SetAttribute ("PacketSize", UintegerValue
(1000));

onOffHelper.SetAttribute ("DataRate", StringValue
("1000bps"));

onOffHelper.SetAttribute ("StartTime", TimeValue (Seconds
(1.000000)));

cbrApps.Add (onOffHelper.Install (nodes.Get (1)));
```

Pakety sú generované s intervalom PacketSize/DataRate.

VBR

Poissonove rozdelenie:

$$P(N_t = k) = e^{-\lambda t} \frac{(\lambda t)^k}{k!}$$

Stredná hodnota = λ, rozptyl = λ, kde λ je parameter.

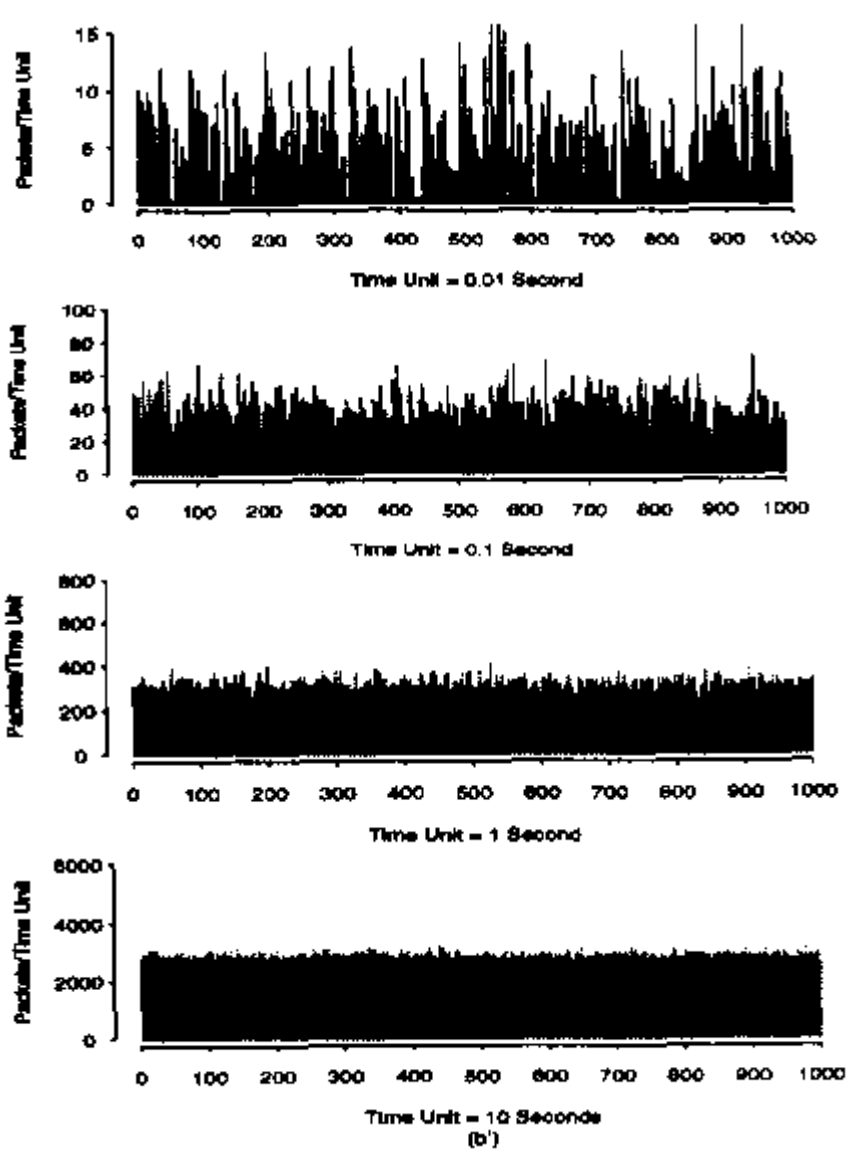
Pre k = 0, 1, 2, 3, ...

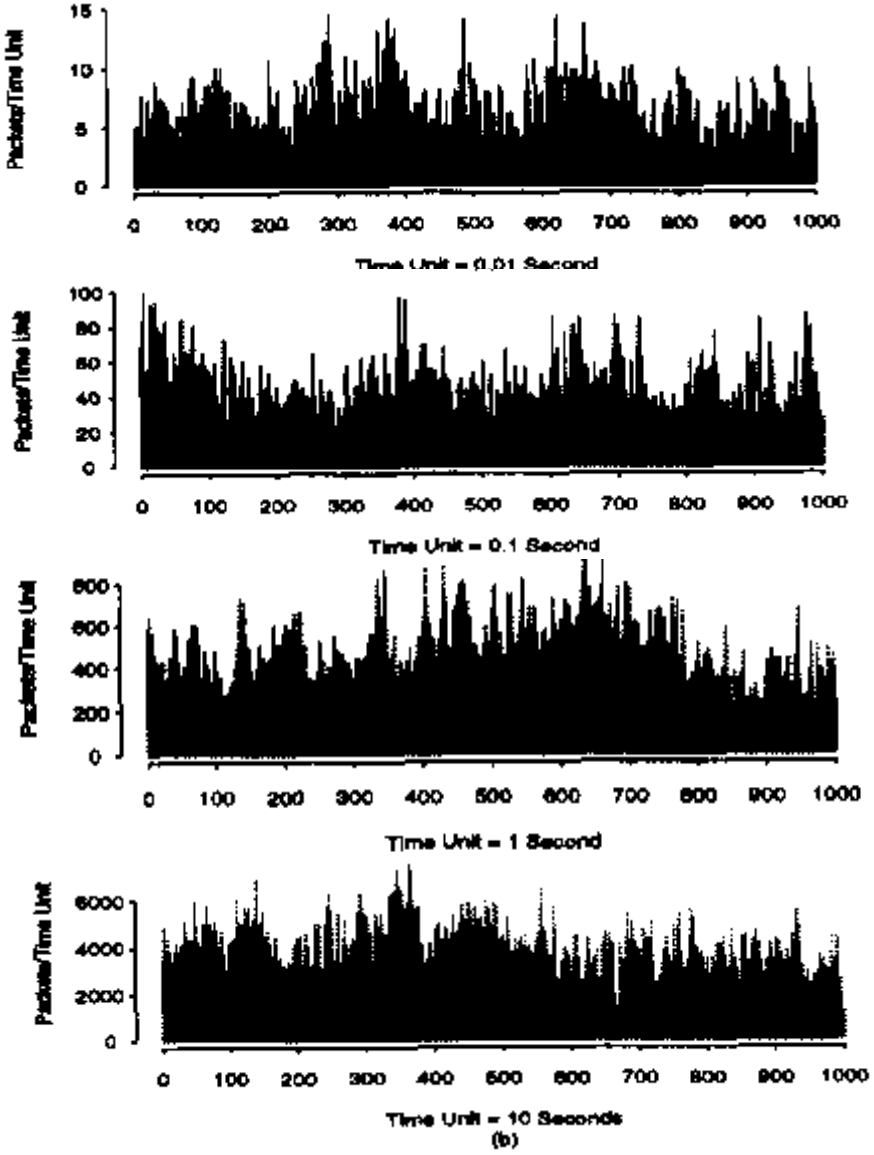
Pravdepodobnosť výskytu udalosti do času t = k.

Interval medzi udalosťami má v Poissonovom procese exponenciálne rozdelenie.

$$P(N_t = 0) = e^{-\lambda t}$$

Poisson vs realita





Experiment

Dáta z Ethernetovej siete z rokov *1989-1992*.
DEC 8650 minicomputers, SPARCstation 1...

Poissonove rozdelenie má strednú hodnotu λ, po výpočte priemernej hodnoty nameraných vzoriek konverguje táto hodnota k λ.

To je možné vidieť na ľavej strane predchádzajúceho obrázku, keď napr. pri priemere za 10s je možné pozorovať ako sa rôznorodosť stráca, pričom na y-osi je zobrazené oneskorenie dátového paketa od predchádzajúceho paketa.

V Poissonovom procese sú udalosti na sebe nezávislé. Realita je zobrazená v pravej časti obrázku. Ukázalo sa, že vzájomné oneskorenie dátových paketov nie je možné modelovať ako nezávislú udalosť.

Napriek nedostatku Poissonovho procesu je tento model najviac používaný, pretože je jasne interpretovateľný.

Poissonove rozdelenie vs realita

Poisson: čas udalostí je nezávislý

Realita: udalosti sú závislé (korelované)

$$r(k)=\frac{E\left[\left(x_t-\overline{x}\right)\left(x_{t+k}-\overline{x}\right)\right]}{s^2}$$; autokorelačná funkcia

$$r(0)=\frac{E\left[\left(x_t-\overline{x}\right)\left(x_t-\overline{x}\right)\right]}{s^2}=1$$

ak x_t, x_{t+k} sú identicky rozdelené a nezávislé potom platí:

$$r(k)=\frac{E\left[\left(x_t-\overline{x}\right)\left(x_{t+k}-\overline{x}\right)\right]}{s^2}=\frac{E\left[\left(x_t-\overline{x}\right)\right]E\left[\left(x_{t+k}-\overline{x}\right)\right]}{s^2}=0$$

Realita

Autokorelačná funkcia r(k) má v mnohých prípadoch tvar:

$$r(k) \sim k^{-b}$$

$$H = 1 - b/2$$

b má súvis s Hurstovým parametrom H, ktorý je mierou dlhodobej závislosti v časovej sérii.

Autokorelácia sa znižuje s rastúcim intervalom výskytu udalostí.

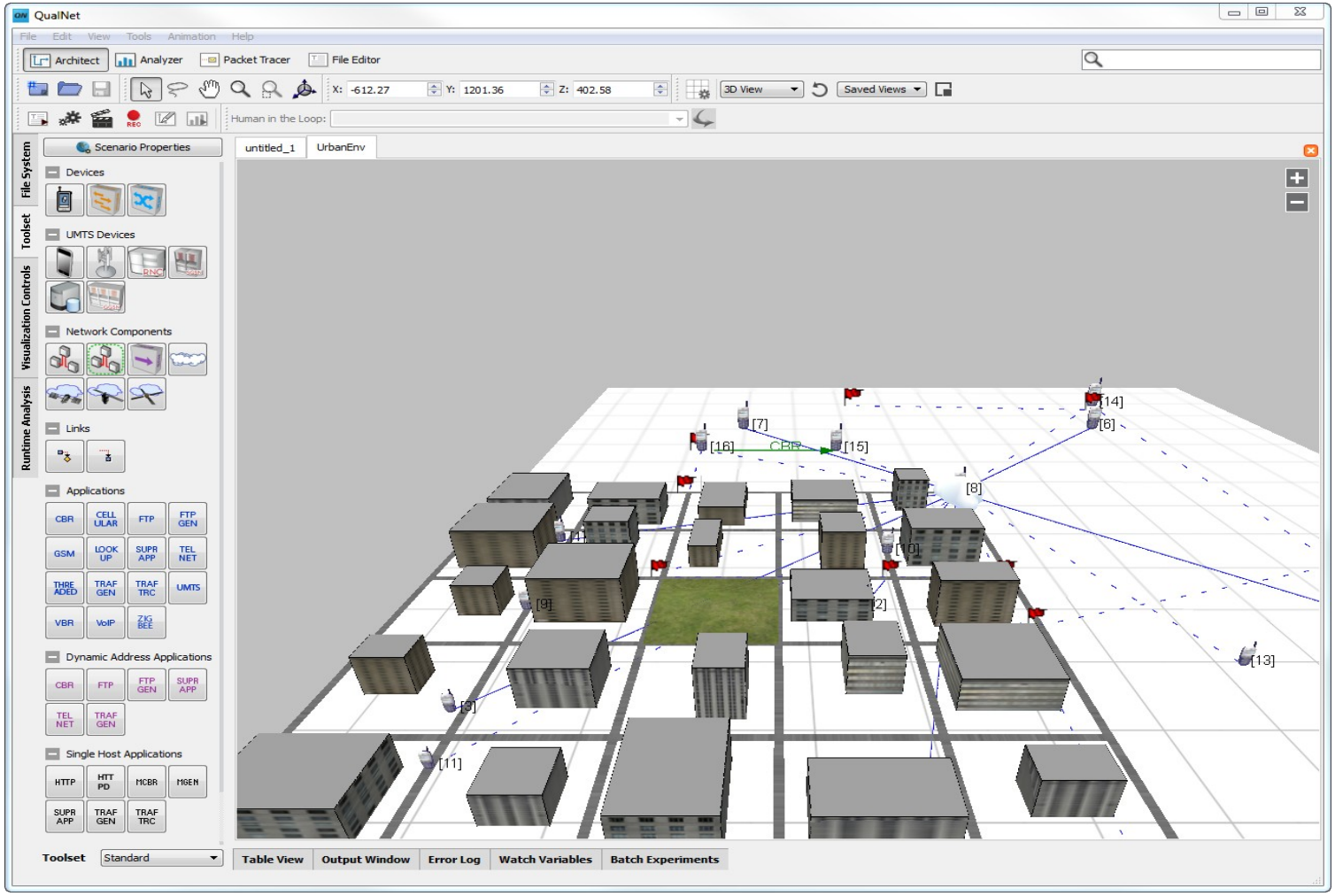
Brownov pohyb má H = 0.5, bez autokorelácie.

0.5 < H < 1.0, proces s dlhodobou pozitívnou koreláciou.

Simulačné nástroje

Glomosim (UCLA)– Qualnet: Parsec (podobné jazyku C)
ns2 (USC): C++/OTcl
ns3: C++
JiST/swans (Cornell): Java
OPNET
OMNet++

Qualnet



Niektoré simulačné nástroje umožňujú modelovanie reálneho prostredia. Realistické modely miest je možné zakúpiť.

Simulácia je...

Technika imitácie správania sa systémov pomocou analogických modelov, situácii a aparátu s cieľom získať informáciu alebo trénovať ľudí.

"The technique of imitating the behaviour of some situation or system (economic, mechanical, etc.) by means of an analogous model, situation, or apparatus, either to gain information more conveniently or to train personnel."

(from the Oxford dictionary)

Systémy

- *Spojité systémy*

Zmena nastáva spojite

- *Diskrétne systémy*

Zmena nastáva v dopredu určených okamžikoch

- *Udalostné systémy*

Zmena nastáva po (náhodných) udalostiach...a teda po zmenách stavu

Posun času

• V pravidelných časových intervaloch t_0, t_1, t_2, . . . (synchrónna simulácia). Nevýhoda: intervaly môžu byť krátke.

• Po udalostiach (asynchrónna simulácia), každá udalosť môže spôsobiť zmenu stavu. Asynchrónna simulácia je štandardne použitá v sieťových simulátoroch.

Terminológia

- *Systém*

Množina objektov, ktoré navzájom interagujú (napr. bezdrôtová sieť).

- *Model*

Matematická reprezentácia systému

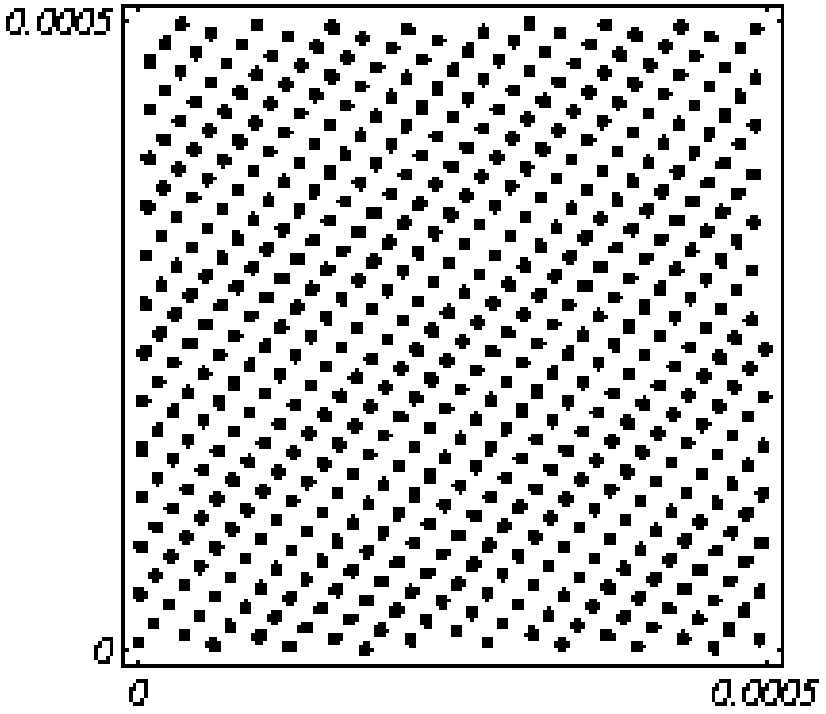
- *Entita*

Objekt systému napr. uzol siete.

- *Atribút*

Vlastnosť entity, napr. počet generovaných paketov.

Moja simulácia je taká dobrá ako...

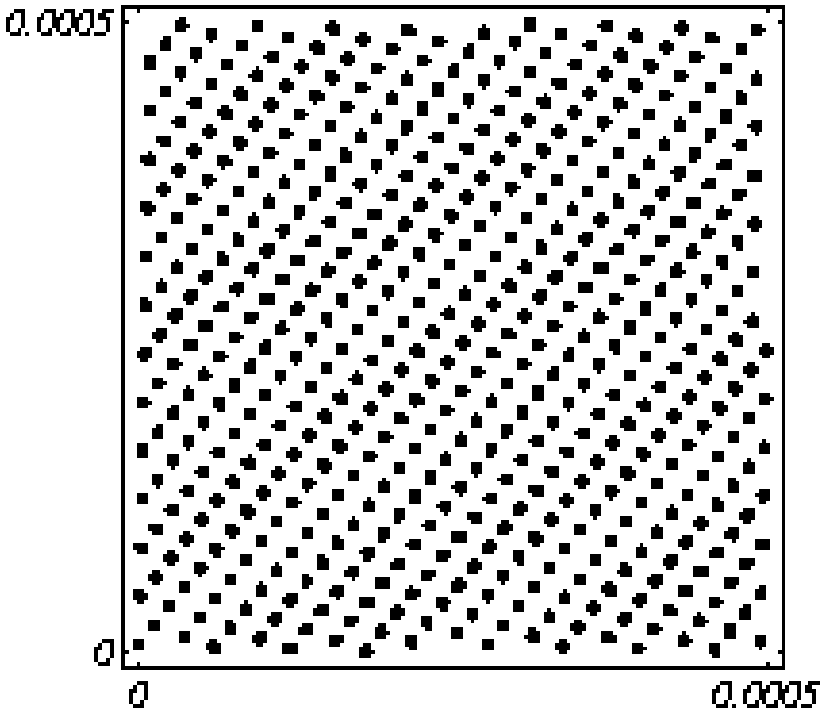


Linear congruential generator (LCG)

$y_{n+1} = a\, y_n + b \pmod m$

ANSI C: *rand()*

a = 1103515245
y_0 = 12345 (*SEED*)
b = 12345
m = 2^31



Moja simulácia je taká dobrá, ako môj generátor náhodných čísel. Všimnime si, že generované čísla po zobrazení všetkých hodnôt vytvárajú vzor. Pri simulácii je veľká spotreba náhodných čísel, často sú potrebné desiatky, stovky miliónov, ...

ns3: MRG32k3a

Pred použitím simulátora je potrebné overiť, aký algoritmus je použitý na generovanie náhodných čísel. ns3 používa MRG32k3a.

Použi:

SeedManager::SetSeed (3);

alebo

SeedManager::SetRun (7);

Vlastnosti:

- Perióda: 10^{57}
- Vlastnosti analyzované teoreticky
- Podpora vláken

ns3: rozdelenia

ns3::UniformVariable

ns3::ExponentialVariable

ns3::NormalVariable

ns3::WeibullVariable

ns3: rozdelenia - príklad

```
#include "ns3/core-module.h"
#include "ns3/ptr.h"
#include <iostream>

using namespace ns3;

int main (int argc, char *argv[]) {
  double min = 0.0;
  double max = 1.0;

  Ptr<UniformRandomVariable> x0 =
CreateObject<UniformRandomVariable>();
  x0->SetAttribute("Min", DoubleValue (min));
  x0->SetAttribute("Max", DoubleValue (max));

  for(int i = 0; i < 20; ++i) {
    double rnd = x0->GetValue();
    std::cout << rnd << std::endl;
  }
```

ns3: rozdelenia - príklad

```
  double mean = 3.14;
  Ptr<ExponentialRandomVariable> x1 =
CreateObject<ExponentialRandomVariable>();

  x1->SetAttribute ("Mean", DoubleValue (mean));

  for(int i = 0; i < 20; ++i) {
    double rnd = x1->GetValue();
    std::cout << rnd << std::endl;
  }

  return 0;
}
```

Weibullove rozdelenie

$$f(x;\lambda,k) = \begin{cases} \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k} & x \geq 0, \\ 0 & x < 0, \end{cases}$$

Weibullove rozdelenie sa používa na modelovanie zlyhania (zariadení ako napr. senzorov)

k < 1 : početnosť zlyhaní sa znižuje s časom, teda najviac zlyhaní je na začiatku (Lindy efekt).

k = 0 : početnosť zlyhaní je konštantná.

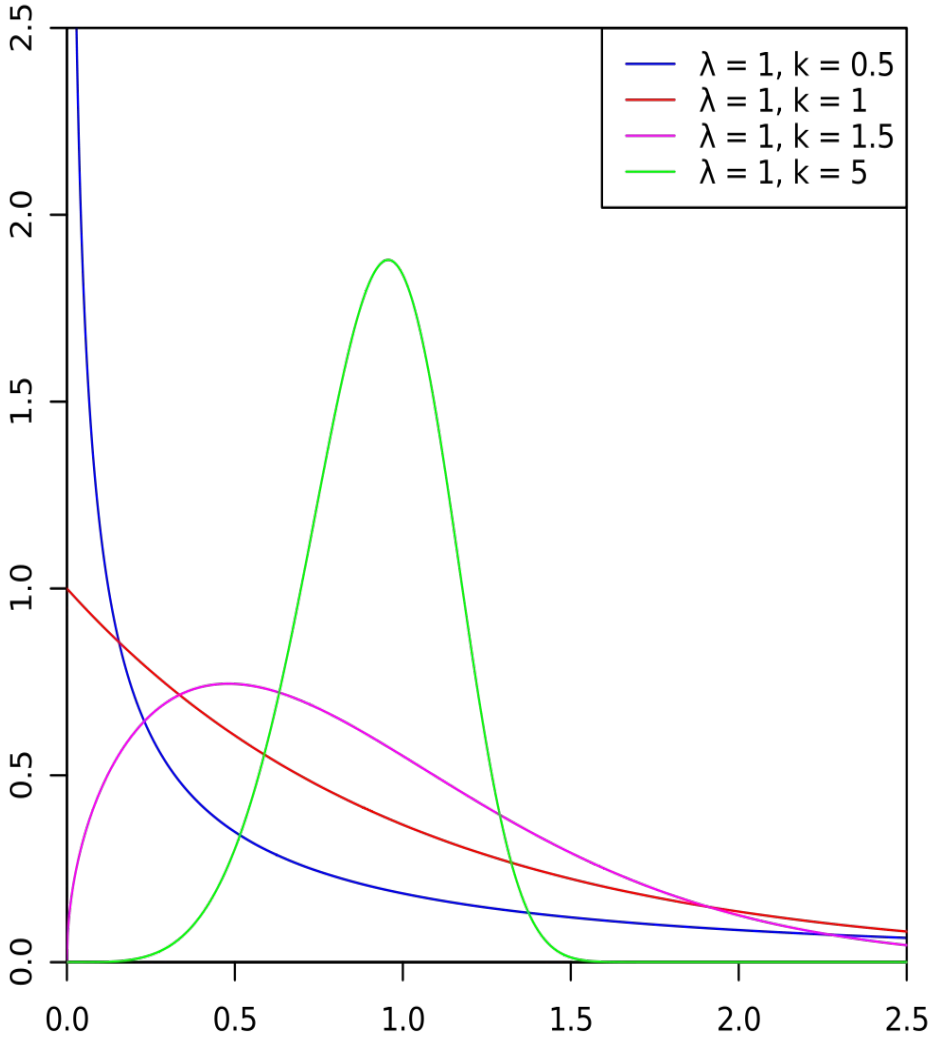
k > 1 : početnosť zlyhaní sa zvyšuje s časom, teda modeluje bežnú chybovosť zariadení.

k = 1 : Weibullove rozdelenie je zovšeobecnenie exponenciálneho rozdelenia

k = 2: Weibullove rozdelenie je zovšeobecnenie Rayleighovo rozdelenia

λ je počet zlyhaní za jednotku času.

Weibullove rozdelenie



Obr. zdroj: Wikipédia, https://en.wikipedia.org/wiki/Weibull_distribution

Úloha

V ns3 namodeluj zlyhávanie uzlov podľa Weibullovho rozdelenia.

Príklad:

```
double scale = 5.0;
double shape = 1.0;
Ptr<WeibullRandomVariable> x =
CreateObject<WeibullRandomVariable> ();
x->SetAttribute ("Scale", DoubleValue (scale));
x->SetAttribute ("Shape", DoubleValue (shape));
```

Simulačné nástroje

Kvalita (vernosť simulácie) vs rýchlosť

Kvalita:
– Simuluje všetky vrtsvy OSI ref. modelu?
– Sú protokoly zjednodušené?

Čas a priestor:
– Rýchlosť: reálny čas
– Priestor: škálovanie s počtom simulovaných uzlov

Dátové štruktúry pre šírenie signálu

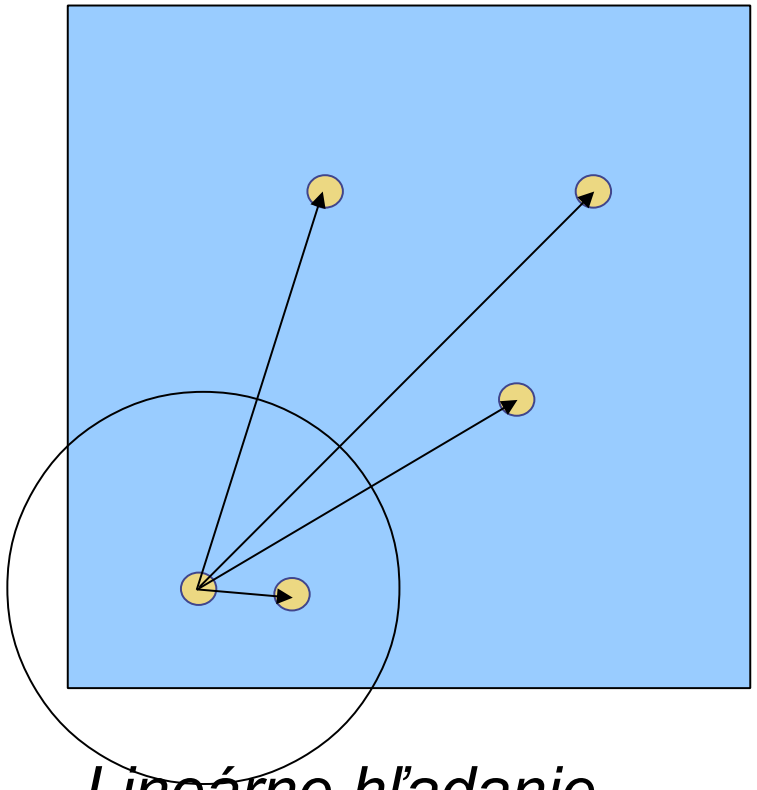
Lineárne hľadanie (O(n)):
Nájdi všetkých susedov a zisti, či sú dosiahnuteľní.

Flat binning:
Simulovaná oblasť je pravidelne rozdelená na štvorce. Hľadaj vo štvorcoch, ktoré sú do určitej vzdialenosti. Vyžaduje sa vedomosť o príslušenstve k štvorcu.

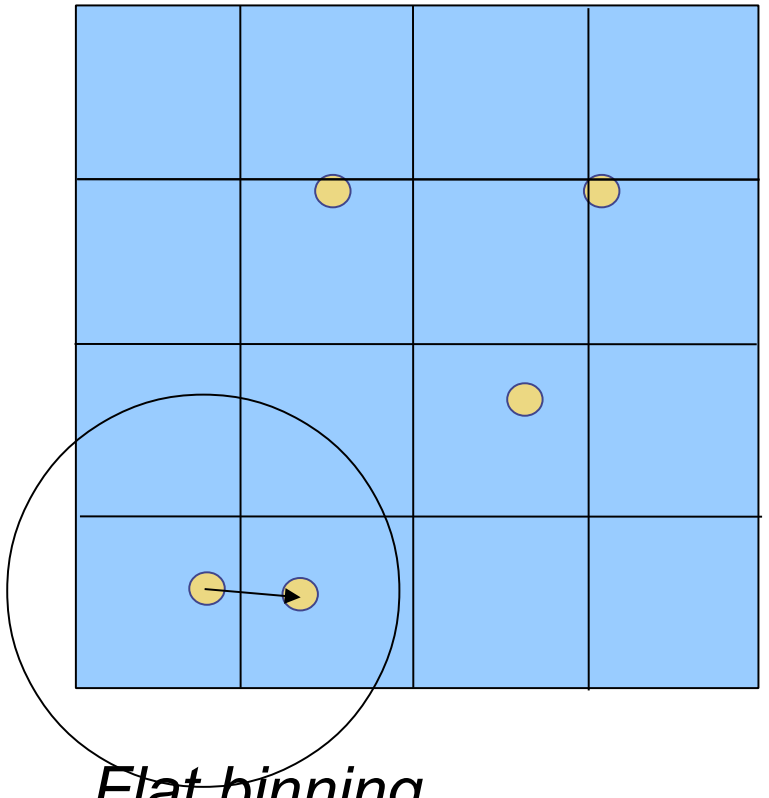
Hierarchické hľadanie:
Rekurzívne rozdeľ simulovanú oblasť pozdĺž osi x a y

Predpoklady: uniformne rozdelené uzly (v jednom štvorci relatívne nízky počet uzlov), akceptovateľná zmena polohy uzlov

Dátové štruktúry pre šírenie signálu

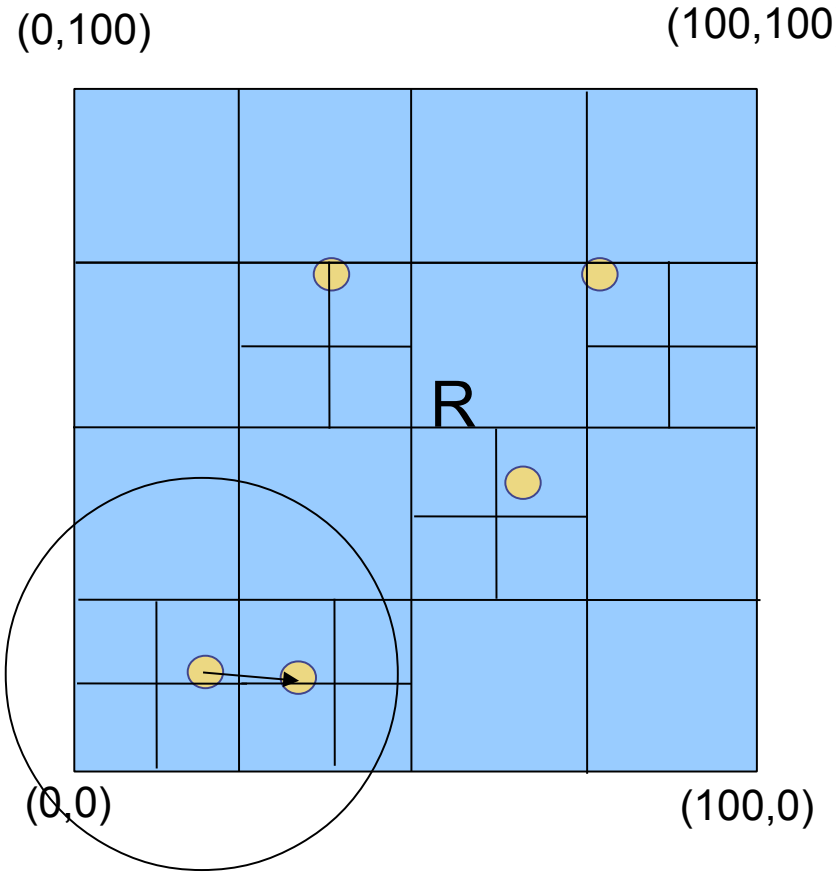


Lineárne hľadanie
Vzdialenosť počítaná pre každý uzol



Flat binning
Vzdialenosť počítaná pre uzly v susedných štvorcoch. Príslušnosť uzla ku štvorcu je potrebná.

Hierarchické hľadanie



Priestorový dekompozičný strom. Deliace body sú fixné koordináty. Štvorce sú listy.

height = log_4(field_size/bin_size)

Príslušnosť ku štvorcu je potrebná.

Začneme v koreni R a delíme priestor rekurzívne. Podobné ako binárne hľadanie, ale v 2D. Je stred štvorca v dosahu signálu?

ns3

ns-3 is a free, open source software project building and maintaining a discrete-event network simulator for research and education

Simulácie sú implementované v C++

Používa Doxygen

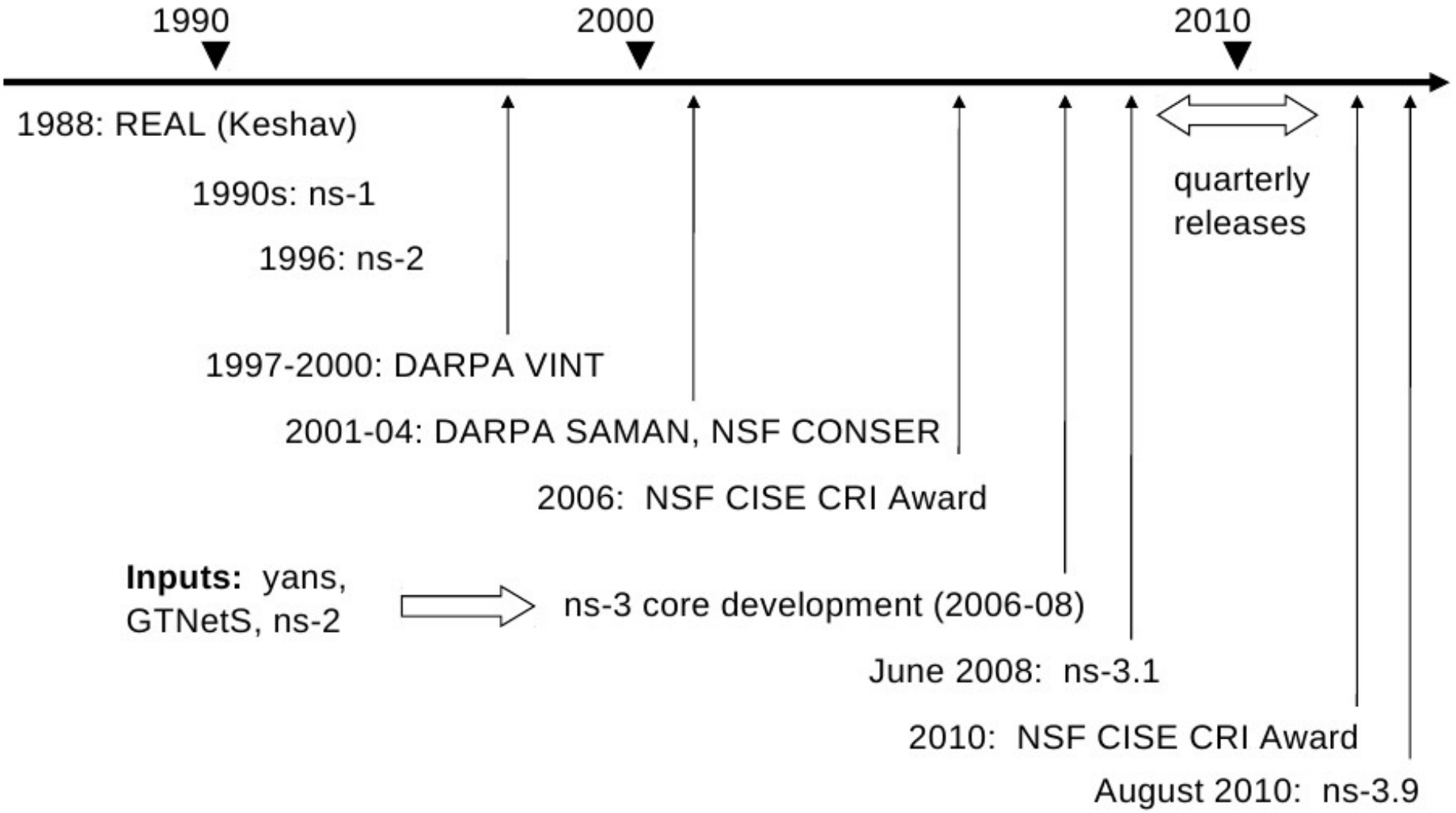
ns-3 má GNU GPLv2 licenciu

Používa *waf:* konfiguračný framework používajúci Python

http://code.google.com/p/waf/

Žiadna spätná kompatibility s ns2 (ns2 používal Tcl pre implementáciu simulácií)

Časová os: ns3



ns-3 : terajšia verzia je 3.30 vydaná 08/2019

Obr. zdroj: ns-3 tutorial at 9th GENI Engineering Conference (GEC9), 2010

Simulátory

REAL (Keshav) : Cornell university, 1997, jazyk C

ns1 : Lawrence Berkeley National Laboratory (LBNL),1995-97

ns2: DARPA, 1997-2000

ns3: National Science Foundation (NSF), 2006-teraz

OPNET: 1986, komečný nástroj

Glomosim: UCLA, 1998, jazyk Parsec.

Qualnet: komerčná verzia Glomosim-u

JiST/swans: Cornell university, 2005, Java

ns3: architektúra

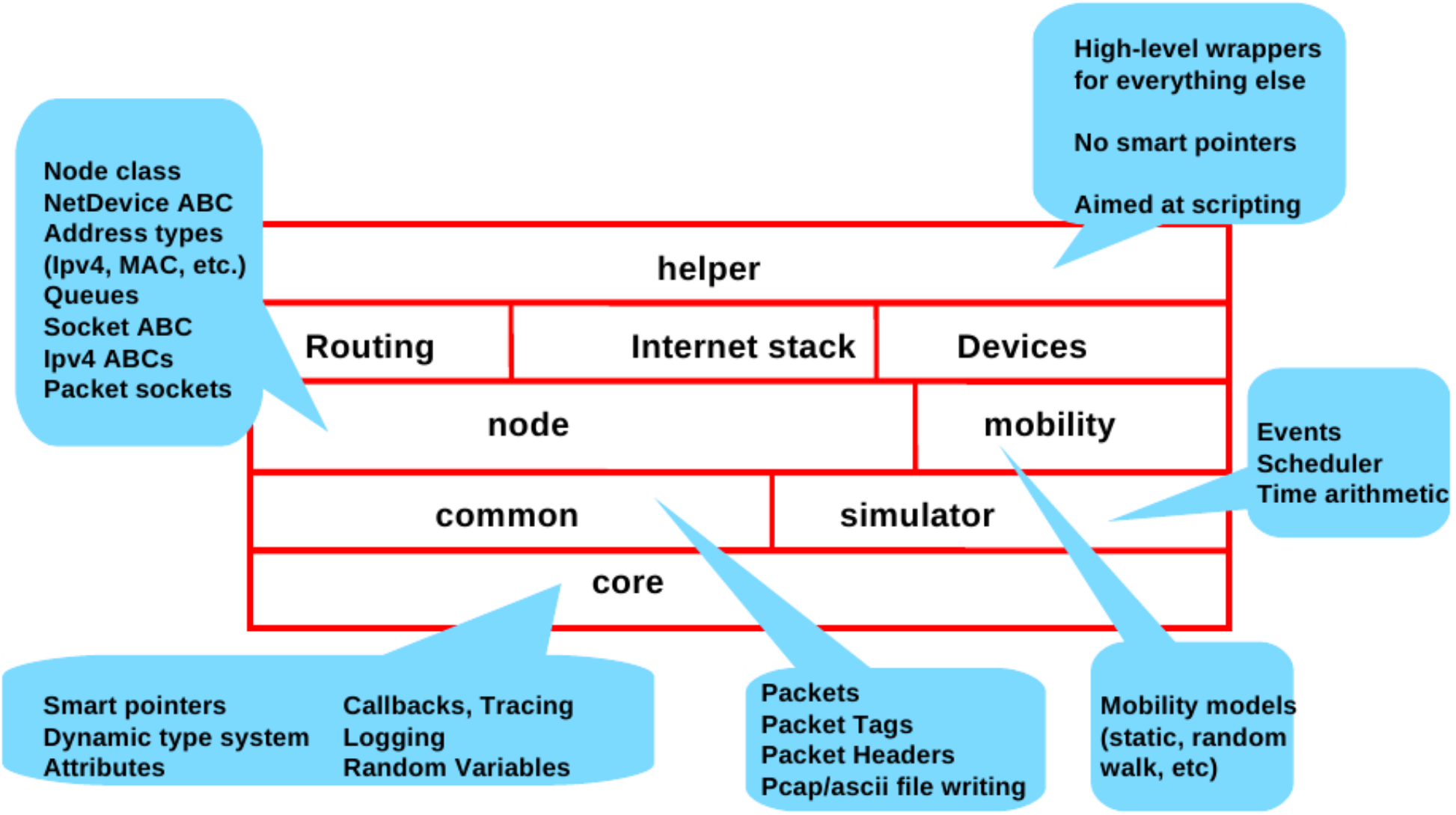


Figure source: ns-3 tutorial at 9th GENI Engineering Conference (GEC9), 2010

ns3

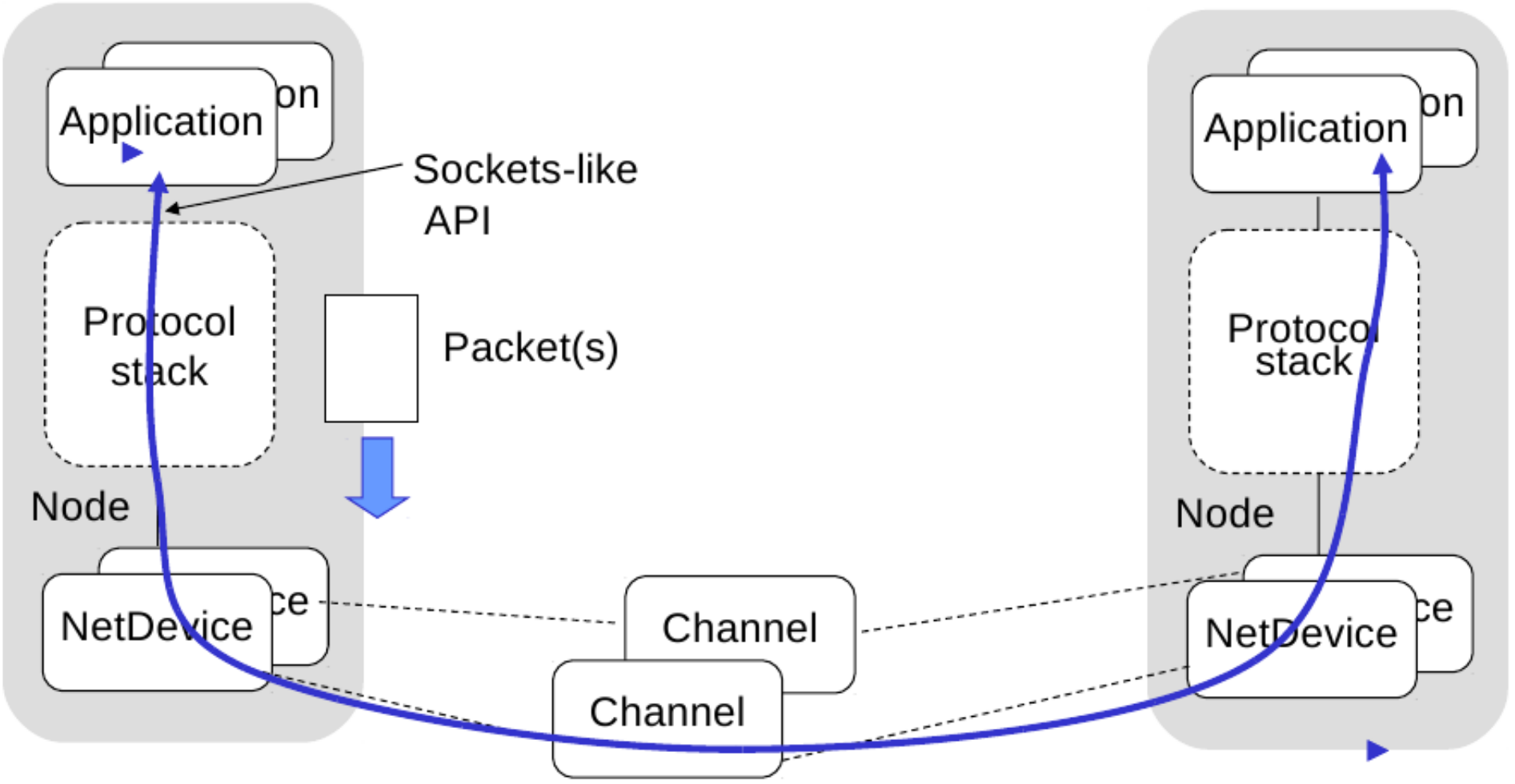


Figure source: ns-3 tutorial at 9th GENI Engineering Conference (GEC9), 2010

ns3

Ns3 obsahuje modely aplikácií, protokolov a prenosu dát.

Dátový paket je vygenerovaný (ľavá strana), pričom môže byť použitý napr. Poissonov proces, následovne je modelovaný jeho prechod cez rôzne protokoly a následne je modelované jeho poslanie cez presnosový kanál.

Dátový paket je prijatý (pravá strana), následne sú aplikované modely použitých protokolov, dátový paket je nakoniec spracovaný v aplikácii.

ns3

```
int main (int argc, char *argv[])
{

  // Set default attribute values

  // Parse command-line arguments

  // Configure the topology; nodes, channels, devices, mobility

  // Add (Internet) stack to nodes

  // Configure IP addressing and routing

  // Add and configure applications

  // Configure tracing

  // Run simulation
}
```

„./waf –run scratch/filename“

Zdroj: ns-3 tutorial at 9th GENI Engineering Conference (GEC9), 2010

ns3

Základná štruktúra ns3 simulácie:

- Konfigurácia topológie
- Konfigurácia protokolov
- Konfigurácia pohybu
- Vznik uzlov
- Konfigurácia výstupu simulácie
- Spustenie simulácie

ns3 : výstup simulácie (tracing)

ASCII:

- Výstup je ASCII súbor

Pcap:

- Výstup pre tcpdump a wireshark

Dokumentácia:

https://www.nsnam.org/docs/manual/html/tracing.html

ns3 : chytré smerníky

```
#include "ns3/core-module.h"
#include "ns3/ptr.h"
#include "ns3/simple-ref-count.h"
#include <iostream>

using namespace ns3;

class Token : public SimpleRefCount<Token> {
public:
  Token() {
    std::cout << "Token()";
  }

  virtual ~Token() {
    std::cout << "~Token()";
  }
};

int main (int argc, char *argv[]) {
  ns3::Ptr<Token> ptr;
  ptr = Create<Token>();

  return 0; }
```

ns3 : chytré smerníky

ns3 využíva „smart pointers“

Trieda SimpleRefCount<T> implementuje manažovanie počtu referencií tak, aby bol na objekt s nulovým počtom referencií zavolaný deštruktor.

Chytrý smerník, pozri kap. 21 (shared pointer):

https://uim.fei.stuba.sk/wp-content/uploads/2018/02/cpp-1.pdf

ns3 : AggregateObject

ns3 využíva využíva spájanie (agregovanie) viacerých objektov

Príklad:

```
//smerník na objekt typu ns3::FriisPropagationLossModel
auto ptr0 = obj.GetObject<ns3::FriisPropagationLossModel>();

//smerník na objekt typu ns3::UanMacAloha
auto ptr1 = obj.GetObject<ns3::UanMacAloha>();
```

Zavolaním GetObject<T>() dostaneme objekt typu T, ktorý je agregovaný s iným objektom. Agregovať je možné len typovo rôzne objekty.

ns3 : AggregateObject

```
#include "ns3/core-module.h"
#include "ns3/ptr.h"
#include "ns3/object.h"
#include "ns3/object-factory.h"
#include "ns3/uan-mac-aloha.h"
#include "ns3/propagation-loss-model.h"

#include <iostream>

using namespace ns3;

int main (int argc, char *argv[]) {

ObjectFactory factory;
factory.SetTypeId("ns3::FriisPropagationLossModel");
ns3::Ptr<ns3::Object> ptrA = factory.Create(); //1. objekt

ObjectFactory factory2;
factory2.SetTypeId("ns3::UanMacAloha");
ns3::Ptr<ns3::Object> ptrB = factory2.Create(); //2. objekt
```

ns3 : AggregateObject

```
ns3::Object obj;

obj.AggregateObject(ptrA); //pridaj 1. objekt
obj.AggregateObject(ptrB); //pridaj 2. objekt

//smerník na objekt typu ns3::FriisPropagationLossModel
auto ptr0 = obj.GetObject<ns3::FriisPropagationLossModel>();

//smerník na objekt typu ns3::UanMacAloha
auto ptr1 = obj.GetObject<ns3::UanMacAloha>();

std::cout << ptr0 << std::endl;
std::cout << ptr1 << std::endl;

ptr0->SetFrequency(1000);

std::cout << ptr0->GetFrequency() << std::endl;

return 0;
}
```

ns3 : AggregateObject

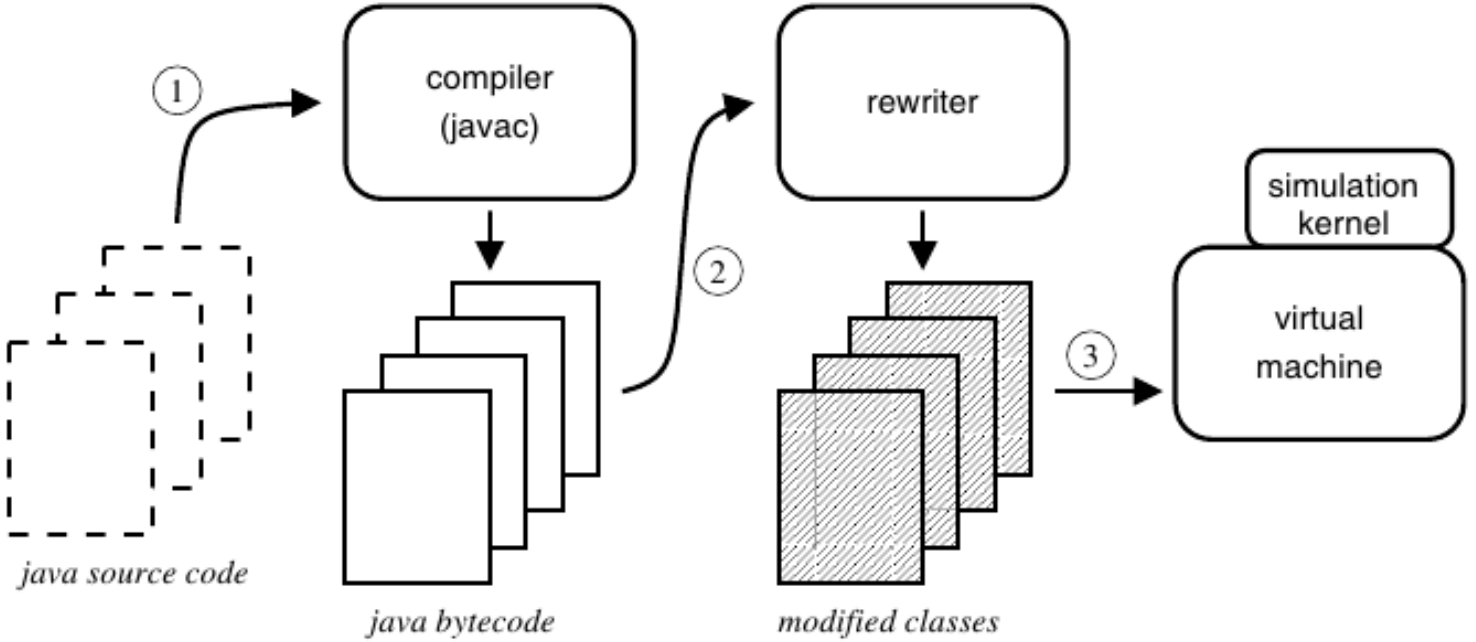
V príklade je tiež využitá ObjectFactory pre zjednodušené generovanie objektov.

Dokumentácia:

https://www.nsnam.org/docs/manual/html/object-model.html

JiST/swans

JiST/swans používal rewriter (byte code transformáciu) pre zvýšenú efektivitu simulácie



„The rewriter component of JiST is a dynamic class loader. It intercepts all class load requests and subsequently verifies and modifies the requested classes. These modified, rewritten classes now incorporate the embedded simulation time operations, but they otherwise completely preserve the existing program logic. The program transformations occur once, at load time, and do not incur rewriting overhead during execution. The rewriter also does not require source-code access, since this is a byte code to byte code transformation.“

Text/figure source: Barr, R. and Haas, Z.J. and van Renesse, R. JiST: an efficient approach to simulation using virtual machines. Software Practice and Experience, vol.35, no. 6, pp. 539-576, 2005.

JiST/swans: škálovatelnosť

JiST/swans používal rewriter (byte code transformáciu) pre zvýšenú efektivitu simulácie, rýchlosť simulácie bola vyššia ako v prípade ns2 a Glomosim, ktoré sú implementované v jazyku C

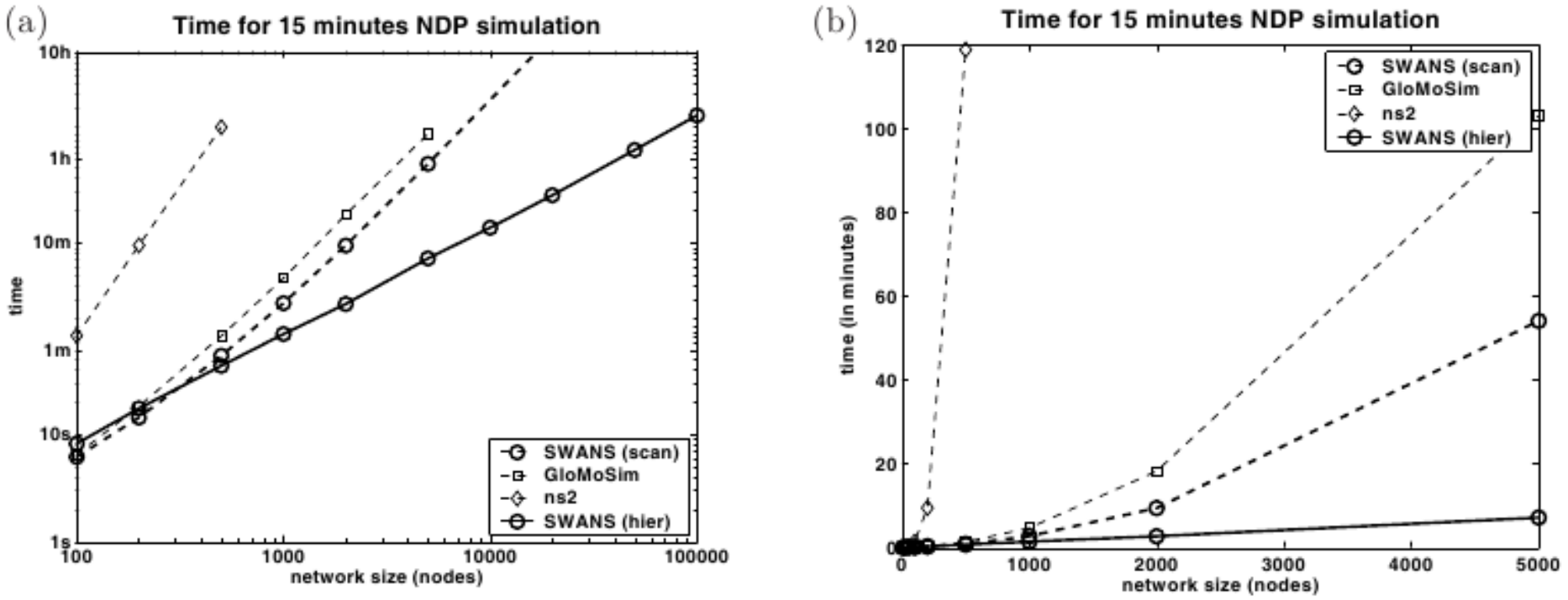


Figure 9. SWANS significantly outperforms both ns2 and GloMoSim in simulations of the node discovery protocol: (a) log–log scale; (b) linear scale.

Zdroj: Barr, R. and Haas, Z.J. and van Renesse, R. JiST: an efficient approach to simulation using virtual machines. Software Practice and Experience, vol.35, no. 6, pp. 539-576, 2005.

Domáce úlohy

ns3 events:

https://www.nsnam.org/docs/manual/html/events.html

ns3 random variables:

https://www.nsnam.org/docs/manual/html/random-variables.html

ns3 object model:

https://www.nsnam.org/docs/manual/html/object-model.html

ns3 tracing:

https://www.nsnam.org/docs/manual/html/tracing.html

Q&A

Q&A budú cez Discord