画像情報特論 (4)
Advanced Image Information (4)

Network Simulators and Emulators

情報理工・情報通信専攻　甲藤二郎

E-Mail: katto@waseda.jp

Network Simulation & Emulation

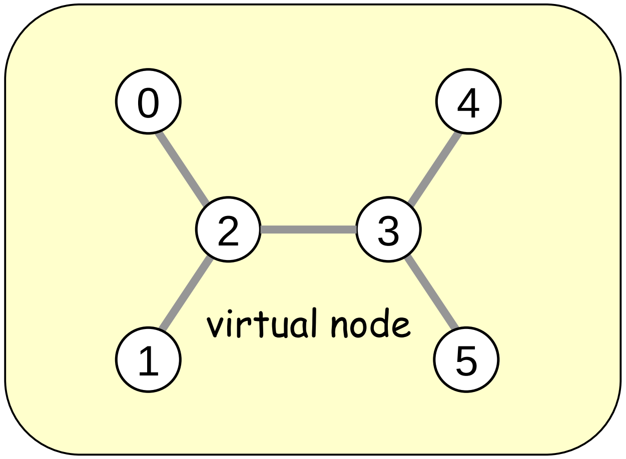
Networking Research

- Algorithm
- Theory (model)
- <u>Simulation</u>
- <u>Emulation</u>
- Implementation (testbed)

Simulator & Emulator (1)

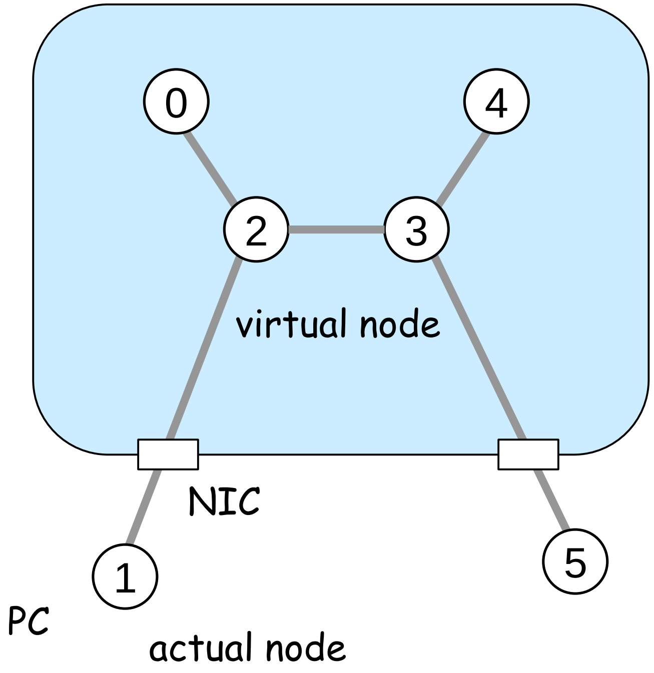
- simulation

software



- emulation

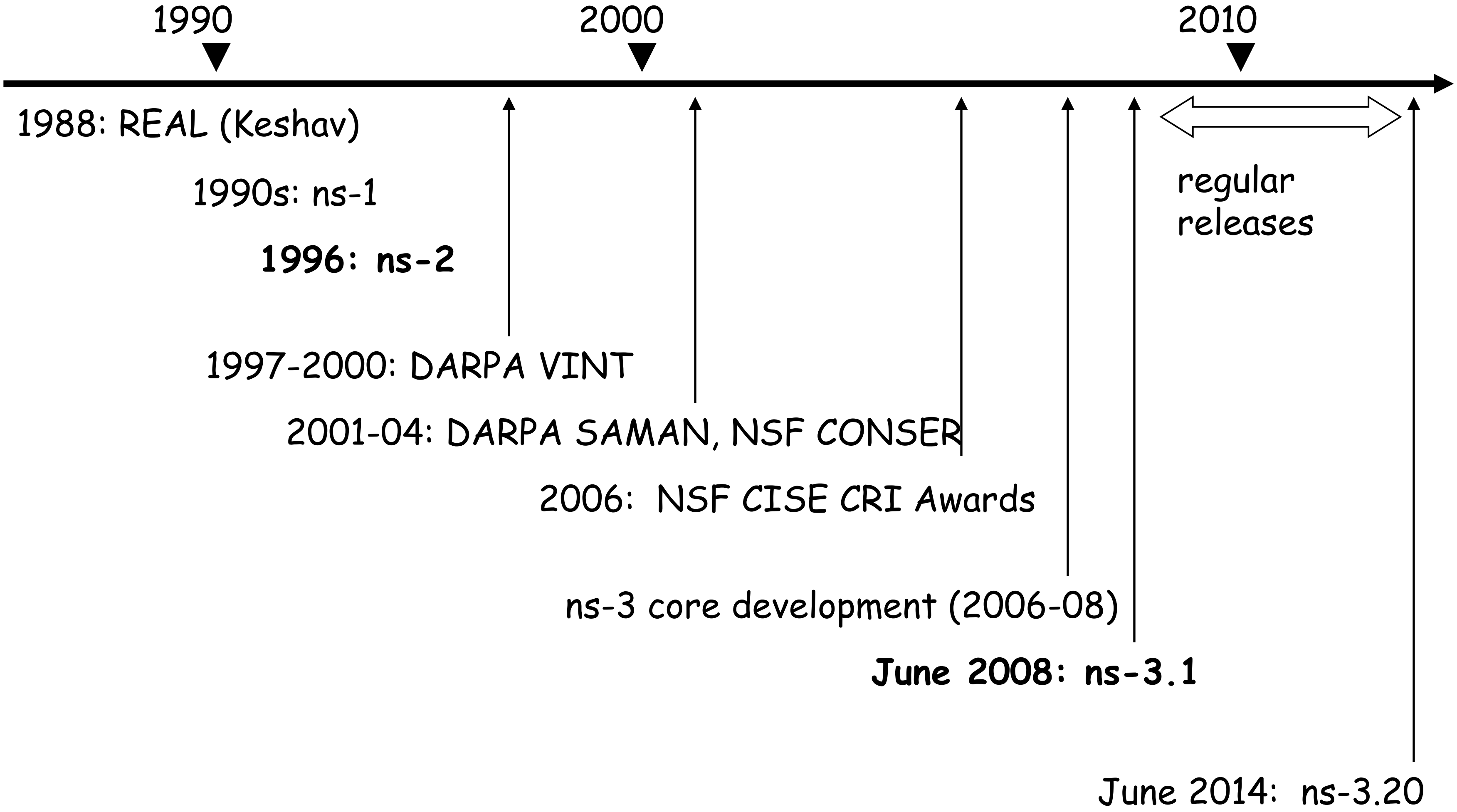
software / hardware



Simulator & Emulator (2)

simulator	emulator	URL
ns-2 (ns)	nse	http://www.isi.edu/nsnam/ns/
ns-3	Emu/Tap device	http://www.nsnam.org/
OPNET		https://www.riverbed.com/jp/products/steelcentral/opnet.html
Qualnet, GloMoSim	EXata	http://web.scalable-networks.com/
	PacketStorm	http://www.packetstorm.com/

History of ns



1988: REAL (Keshav)

1990s: ns-1

1996: ns-2

1997-2000: DARPA VINT

2001-04: DARPA SAMAN, NSF CONSER

2006: NSF CISE CRI Awards

ns-3 core development (2006-08)

June 2008: ns-3.1

regular releases

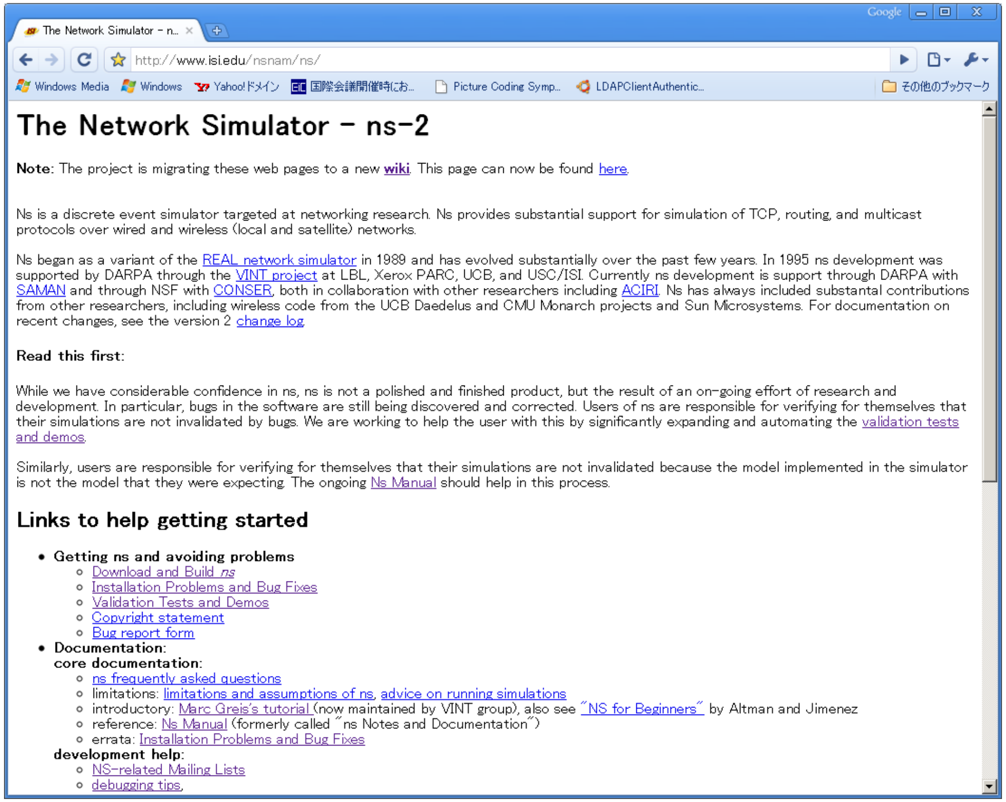
June 2014: ns-3.20

http://www.nsnam.org/docs/ns-3-overview.ppt

ns-2

Ns-2 (1)

- http://www.isi.edu/nsnam/ns/



The Network Simulator - ns-2

Note: The project is migrating these web pages to a new **wiki**. This page can now be found here.

Ns is a discrete event simulator targeted at networking research. Ns provides substantial support for simulation of TCP, routing, and multicast protocols over wired and wireless (local and satellite) networks.

Ns began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is support through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. Ns has always included substantal contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems. For documentation on recent changes, see the version 2 change log.

Read this first:

While we have considerable confidence in ns, ns is not a polished and finished product, but the result of an on-going effort of research and development. In particular, bugs in the software are still being discovered and corrected. Users of ns are responsible for verifying for themselves that their simulations are not invalidated by bugs. We are working to help the user with this by significantly expanding and automating the validation tests and demos.

Similarly, users are responsible for verifying for themselves that their simulations are not invalidated because the model implemented in the simulator is not the model that they were expecting. The ongoing Ns Manual should help in this process.

Links to help getting started

- **Getting ns and avoiding problems**
  - Download and Build *ns*
  - Installation Problems and Bug Fixes
  - Validation Tests and Demos
  - Copyright statement
  - Bug report form
- **Documentation:**

  core documentation:
  - ns frequently asked questions
  - limitations: limitations and assumptions of ns, advice on running simulations
  - introductory: Marc Greis's tutorial (now maintained by VINT group), also see "NS for Beginners" by Altman and Jimenez
  - reference: Ns Manual (formerly called "ns Notes and Documentation")
  - errata: Installation Problems and Bug Fixes

  development help:
  - NS-related Mailing Lists
  - debugging tips,

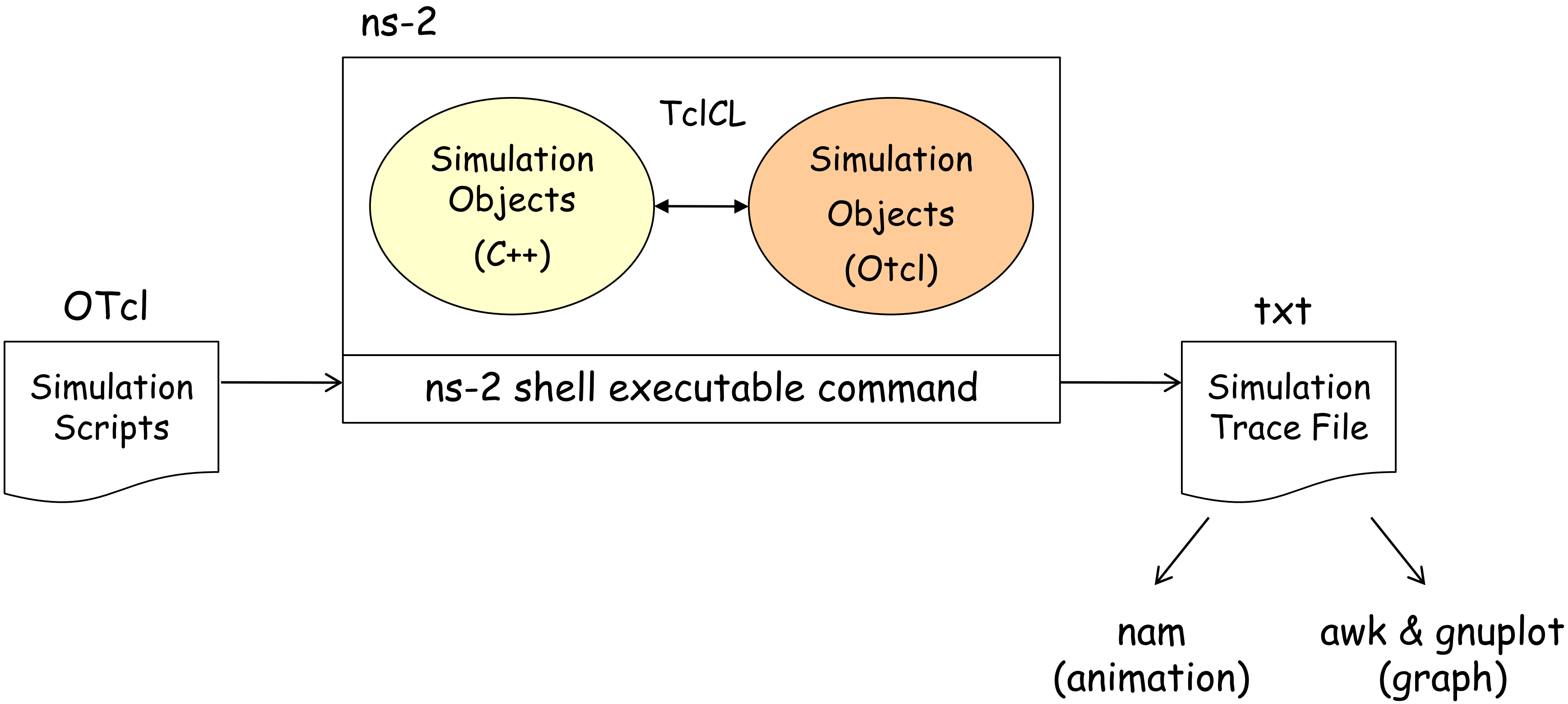
Ns-2 (2)

- download
  - 2.31 and later: http://nsnam.sourceforge.net/wiki/index.php/Main_Page
  - Before 2.31: http://www.isi.edu/nsnam/dist/

Download “allinone”, expand、configure、and make (Tcl/Tk, Otcl, TclCL, ns, nam)

Ns-2 (3)

- ns-2 Architecture



Ns-2 (4)

- Simulation Scripts (*.tcl)

```
# initialization
# Simulator object
set ns [ new Simulator ]
# network topology
# definition of agents and apps
# procedure definition (e.g. finish)
proc finish () …
# event definition
$ns at 1.0 “$ftp start”
# simulation start
$ns run
```

```
set ns [new Simulator]
set f [open out.tr w]
$ns trace-all $f

set n0 [$ns node]
set n1 [$ns node]
$ns duplex-link $n0 $n1 100Mb 1ms DropTail

set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
…
```

Ns-2 (5)

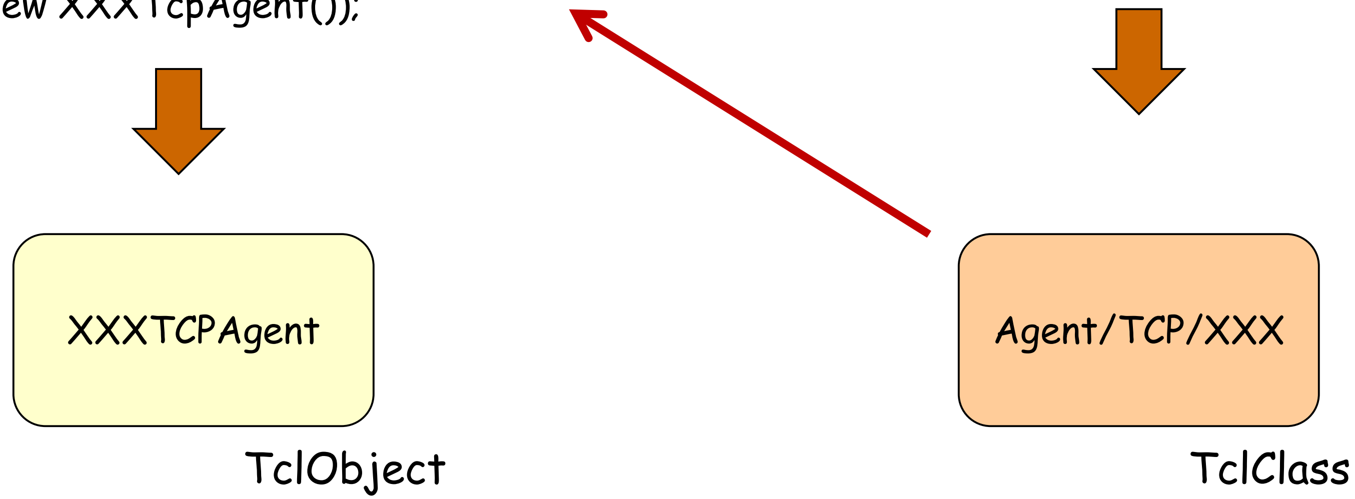
- Simulation Objects (C++/OTcl)

C++

```
static class XXXTcpClass : public TclClass {
public:
   XXXTcpClass() : TclClass("Agent/TCP/XXX") {}
   TclObject* create(int, const char*const*) {
      return (new XXXTcpAgent());
   }
} class_XXX;
```

OTcl

```
set tcp [new Agent/TCP/XXX]
```



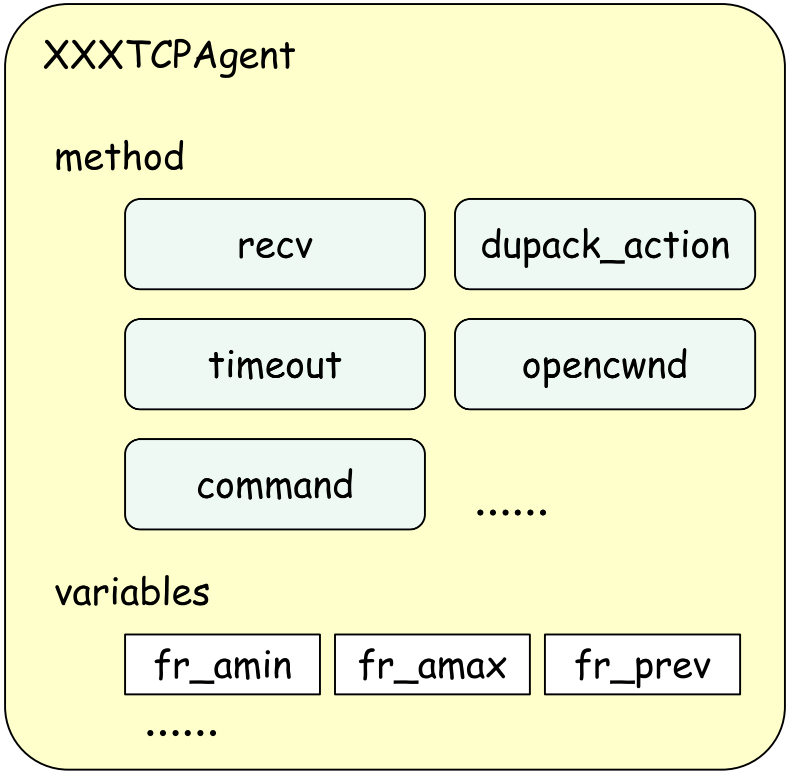
Ns-2 (6)

- Simulation Objects (C++/OTcl)

C++

```
class XXXTcpAgent : public TcpAgent {
public:
   XXXTcpAgent();
   virtual void recv(Packet *pkt, Handler*);
   virtual void dupack_action();
   virtual void timeout (int tno);
   virtual void opencwnd();
   …
protected:
   int command(int argc, const char*const* argv);

   double fr_amin_;
   double fr_amax_;
   double fr_prev_;
}
```



Ns-2 (7)

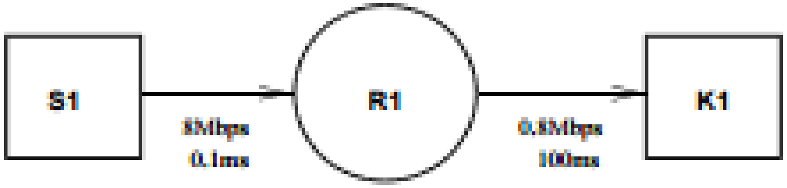
simulation results

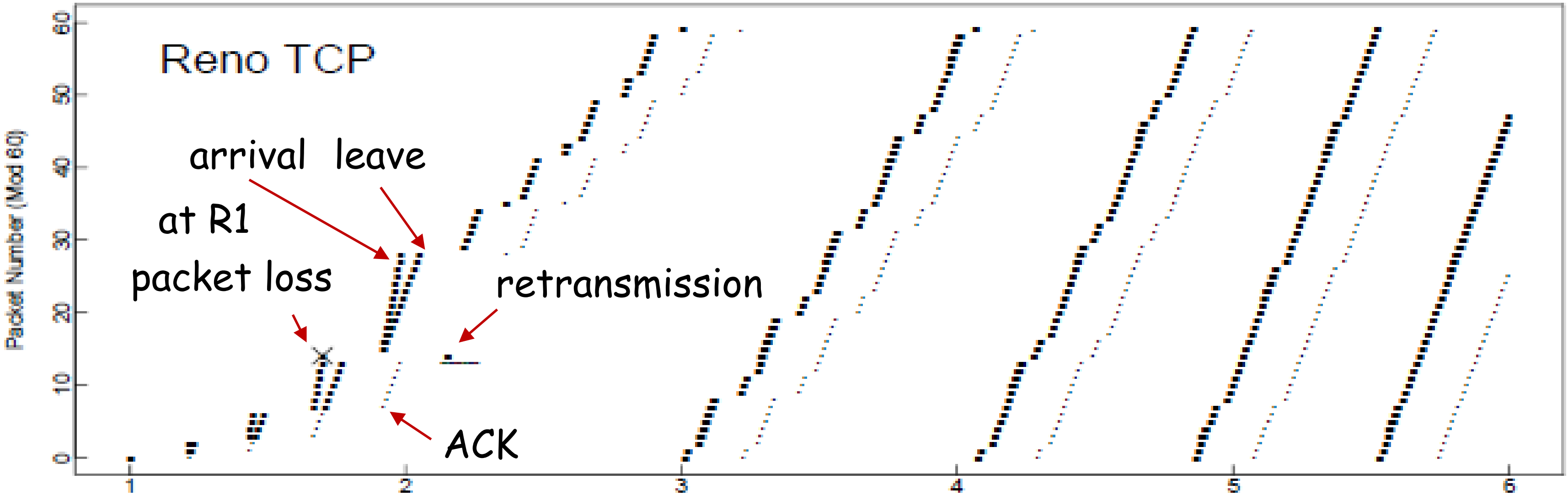
- Trace File (*.tr)

```
enqueue → + 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
dequeue → - 1.84375 0 2 cbr 210 ------- 0 0.0 3.1 225 610
receive → r 1.84471 2 1 cbr 210 ------- 1 3.0 1.0 195 600
          r 1.84566 2 0 ack 40 ------- 2 3.2 0.1 82 602
          + 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
          - 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
          r 1.84609 0 2 cbr 210 ------- 0 0.0 3.1 225 610
          + 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
   drop → d 1.84609 2 3 cbr 210 ------- 0 0.0 3.1 225 610
          - 1.8461 2 3 cbr 210 ------- 0 0.0 3.1 192 511
          r 1.84612 3 2 cbr 210 ------- 1 3.0 1.0 196 603
```

Ns-2 (8)

- example





arrival leave

at R1

packet loss

retransmission

ACK

slow start

congestion avoidance

http://www.icir.org/floyd/papers/sacks.pdf

ns-2 TCP-Linux

http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/index.html

ns-2 TCP-Linux (1)

- ns-2 simulation using TCP implementation code in Linux kernel
  - bridge between implementations (Linux kernel) and simulations (ns-2)
    - fill a gap between implementation and simulation
  - verification of implementation codes

http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/index.html

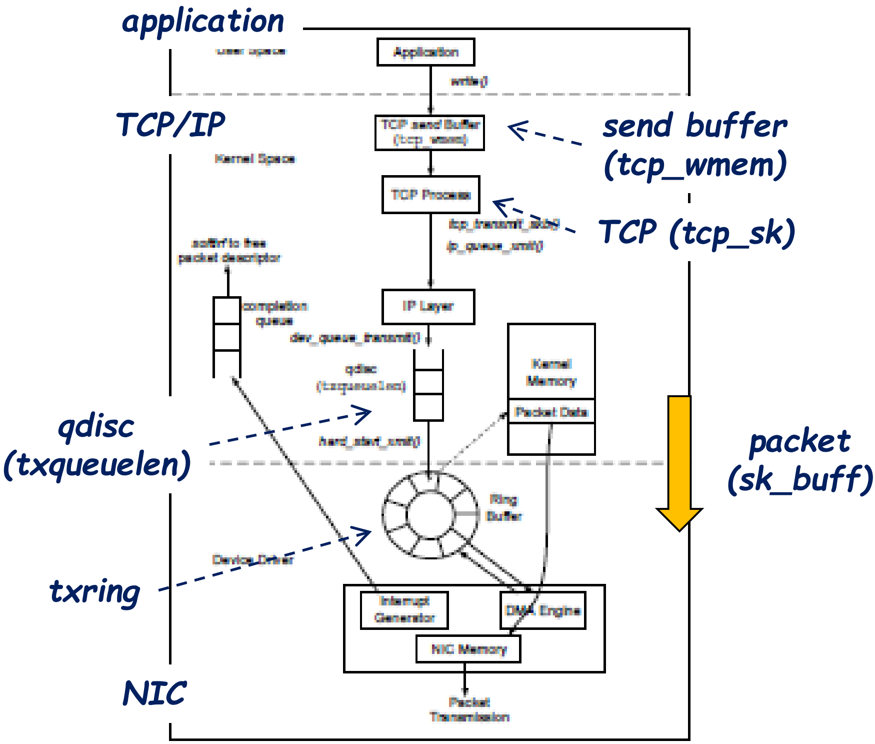
ns-2 TCP-Linux (2)

- TCPs implemented in Linux kernel (2.6.16-3)
  - TCP-Reno, TCP-Vegas, HighSpeed-TCP, Scalable-TCP, BIC-TCP, CUBIC-TCP (Linux), TCP-Westwood, H-TCP, TCP-Hybla, TCP-Veno, TCP-LowPriority, Compound-TCP (Windows), ……

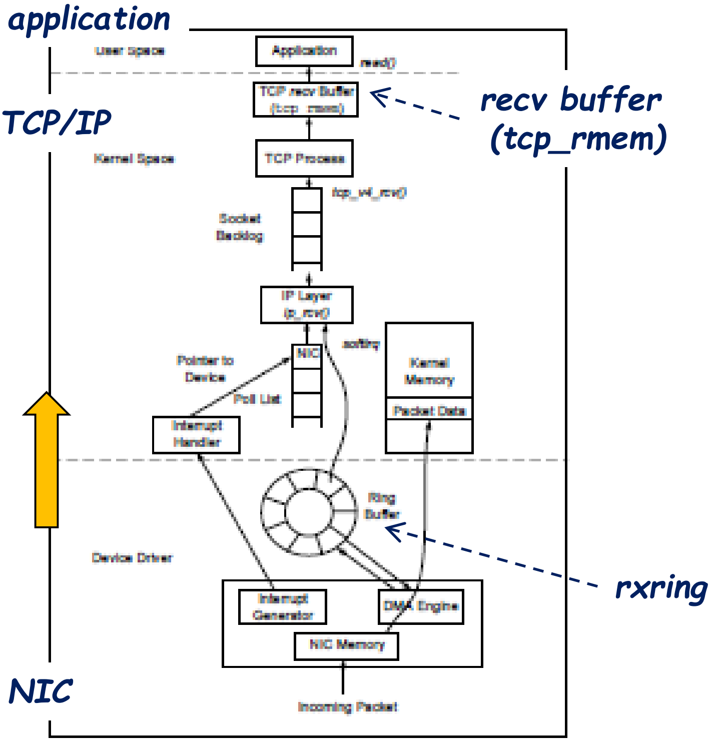
http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/index.html

ns-2 TCP-Linux (3)

- TCP Implementation in Linux



Packet transmission

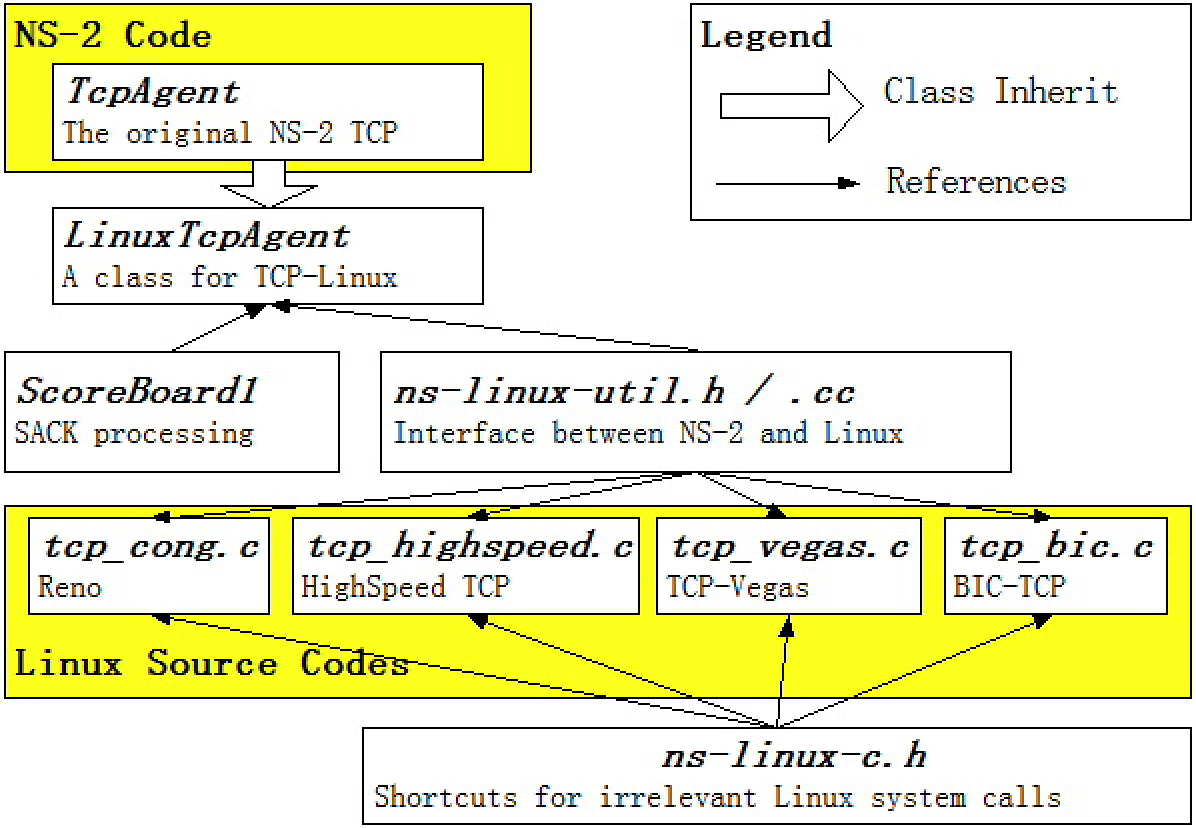


Packet reception

http://www.ece.virginia.edu/cheetah/documents/papers/TCPlinux.pdf

ns-2 TCP-Linux (4)

- Code structure



NS-2 Code

TcpAgent
The original NS-2 TCP

LinuxTcpAgent
A class for TCP-Linux

ScoreBoard1
SACK processing

ns-linux-util.h / .cc
Interface between NS-2 and Linux

tcp_cong.c
Reno

tcp_highspeed.c
HighSpeed TCP

tcp_vegas.c
TCP-Vegas

tcp_bic.c
BIC-TCP

Linux Source Codes

ns-linux-c.h
Shortcuts for irrelevant Linux system calls

Legend

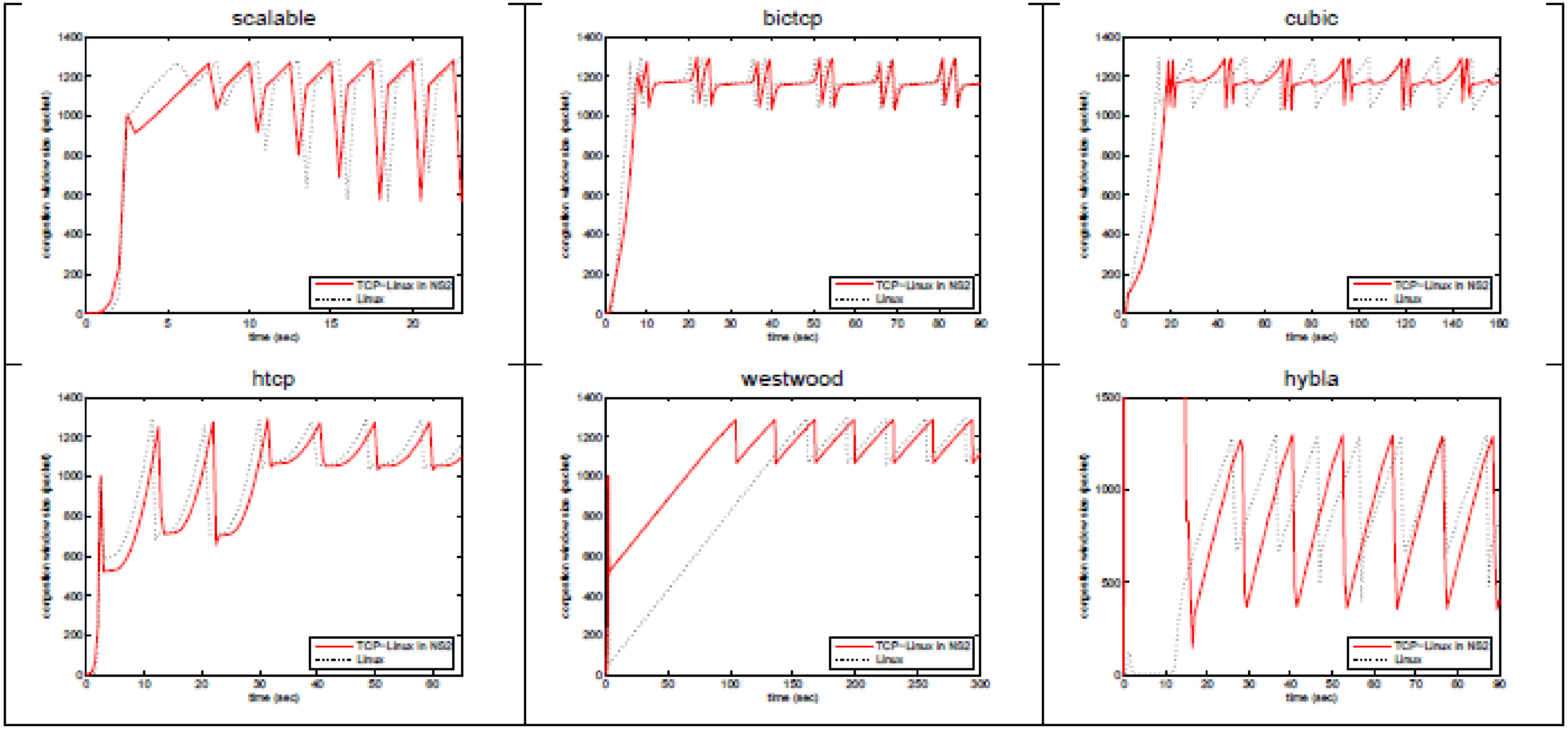
Class Inherit

References

http://netlab.caltech.edu/projects/ns2tcplinux/

ns-2 TCP-Linux (5)

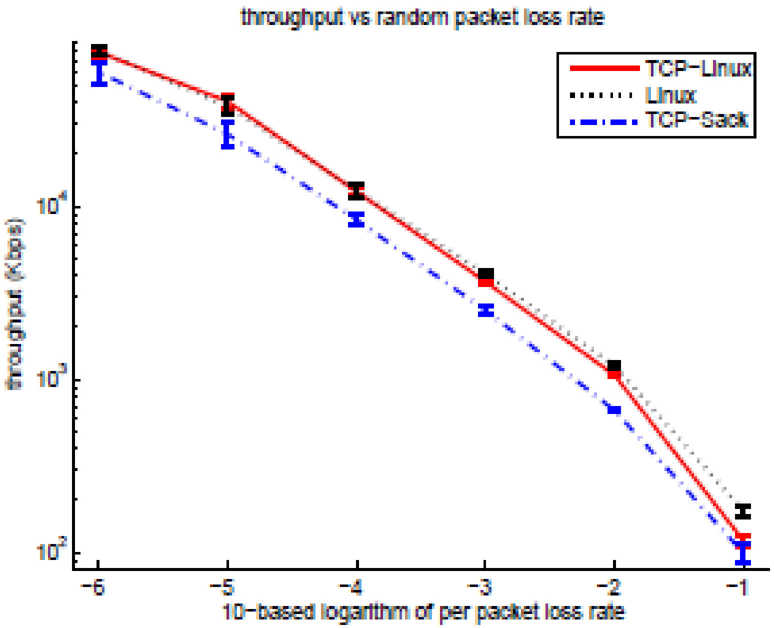
- Simulation (1) ns-2 & Linux



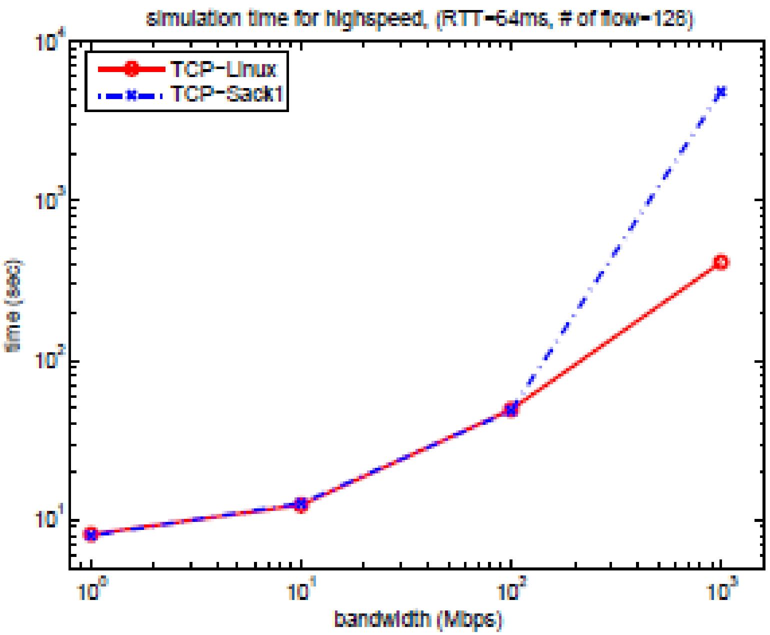
http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/index.html

ns-2 TCP-Linux (6)

- Simulation (2) accuracy & speed



Accuracy

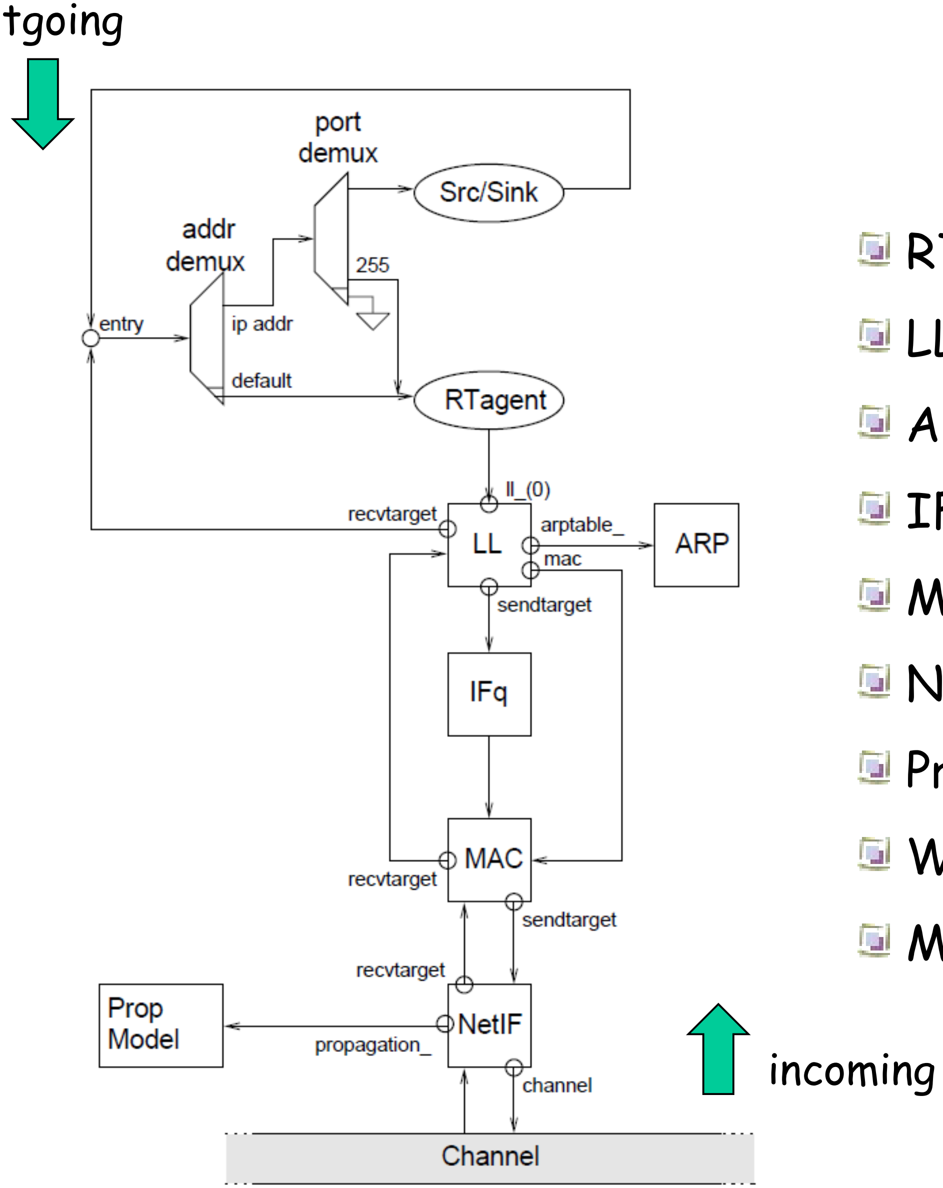


Speed

http://netlab.caltech.edu/projects/ns2tcplinux/ns2linux/index.html

ns-2 wireless model

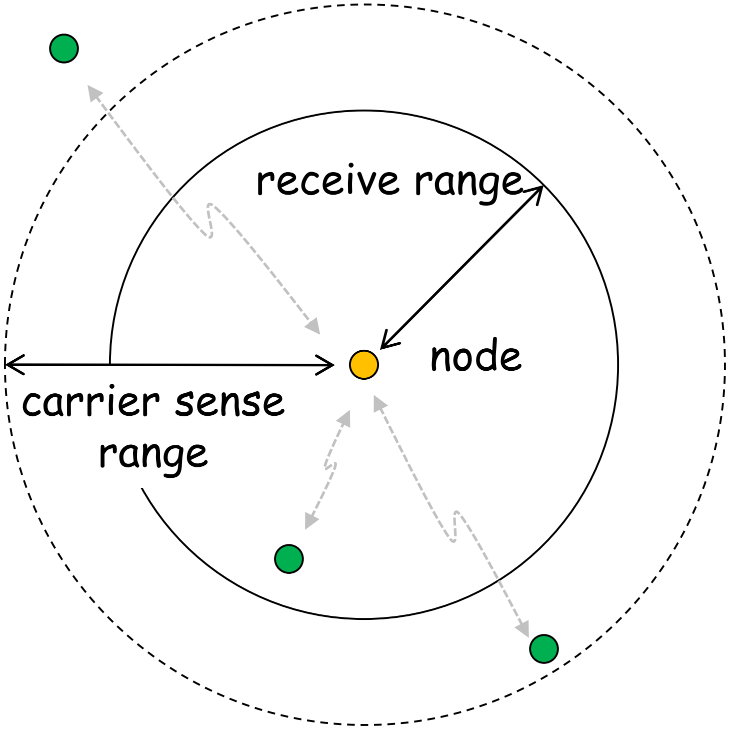
Mobile Node



- RTagent: routing protocol
- LL: link layer
- ARP: ARP table
- IFq: interface queue
- MAC: media access control layer
- NetIF: network interface
- Prop Model: radio propagation
- Wireless Channel
- Mobility Models

CMU Monarch Project: "The CMU Monarch Project's Wireless and Mobility Extensions to ns", Aug.1999.

Radio Propagation (1)



- definition
  - carrier sense range: a node can detect signals
  - receive range: a node can receive packets
  - physical carrier sense: direct signal sensing
  - virtual carrier sense: indirect carrier sensing via RTS/CTS messages

J.Broch et al: "A Performance Comarison of Multi-Hop Ad Hoc Network Routing Protocols", ACM MOBICOM 1998.

Radio Propagation (2)

- parameters
  - Pr: receiving power(function of a distance between nodes)
  - Pr.prev: receiving power of the preceding packets
  - CSThresh: power threshold for carrier sensing
  - RXThresh: power threshold for packet reception
  - CPThresh: power difference which can avoid packet collisions

```
// Network interface
if ( Pr < CSThresh )
  discard as "noise"
elseif ( Pr < RXThresh )
  mark as "error" packet
else
  receive a new packet, goto MAC
  // MAC layer
  if ( state is not "idle" )
    if ( Pr.prev > Pr + CPThresh )
      "capture", drop the new packet
    else
      "collision", drop both packets
  else // ( i.e. state is "idle" )
    receive the new packet
```

http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

Radio Propagation (3)

- Free space model (Friis formula, direct)

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L} \approx \frac{P_t \lambda^2}{(4\pi)^2 d^2}$$

d：distance
Pr(d): receiving power
Pt: transmission power
λ: wavelength

square of distance

- Two-way ground reflection model (direct + reflection)

biquadrate of distance

$$P_r(d) = \frac{P_t G_t G_r h_t^2 h_r^2}{d^4 L} \approx \frac{P_t h_t^2 h_r^2}{d^4}$$

ht：height of transmission antenna
hr: height of receiving antenna

  - near ～ Friis、far ～ Two-ray

cross-over distance:

$$d_c = (4\pi h_t h_r) / \lambda$$

http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

Radio Propagation (4)

- example

tcl/ex/wireless-test.tcl

```
set opt(chan)            Channel/WirelessChannel
set opt(prop)            Propagation/TwoRayGround
set opt(netif)           Phy/WirelessPhy
set opt(ant)             Antenna/OmniAntenna
...
Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 1.5              // height of antenna
Antenna/OmniAntenna set Gt_ 1.0             // transmission gain
Antenna/OmniAntenna set Gr_ 1.0             // reception gain

Phy/WirelessPhy set CPThresh_ 10.0          // capture threshold
Phy/WirelessPhy set CSThresh_ 1.559e-11  // carrier sense threshold (550m)
Phy/WirelessPhy set RXThresh_ 3.652e-10 // packet reception threshold (250m)
Phy/WirelessPhy set Rb_ 2*1e6               // bit rate
Phy/WirelessPhy set Pt_ 0.2818              // transmission power
Phy/WirelessPhy set freq_ 914e+6            // frequency (⇔ wavelength)
Phy/WirelessPhy set L_ 1.0                  // system loss
```

indep-util/propagation/threshold.cc

http://www.isi.edu/nsnam/ns/doc/ns_doc.pdf

ns-3

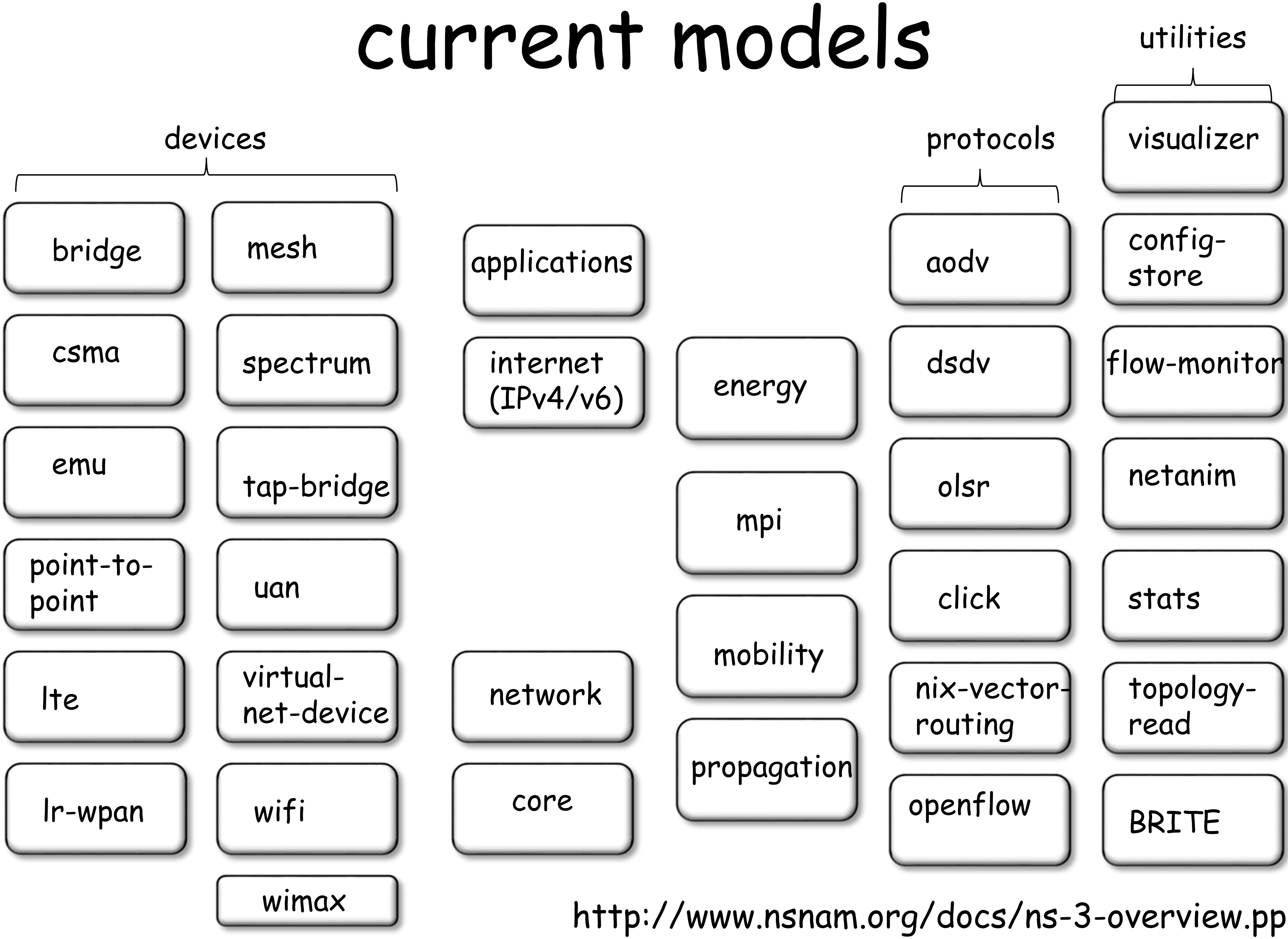
ns-3 software overview

- ns-3 is written in C++, with bindings available for Python
  - simulation programs are C++ executables or Python programs

- ns-3 is a GNU GPLv2-licensed project
- ns-3 is not backwards-compatible with ns-2

http://www.nsnam.org/docs/ns-3-overview.ppt

current models



devices

bridge

csma

emu

point-to-point

lte

lr-wpan

mesh

spectrum

tap-bridge

uan

virtual-net-device

wifi

wimax

applications

internet (IPv4/v6)

network

core

energy

mpi

mobility

propagation

protocols

aodv

dsdv

olsr

click

nix-vector-routing

openflow

utilities

visualizer

config-store

flow-monitor

netanim

stats

topology-read

BRITE

http://www.nsnam.org/docs/ns-3-overview.ppt

relationship to ns-2

Similarities to ns-2:

- C++ software core
- GNU GPLv2 licensing
- ported ns-2 models: random variables, error models, OLSR, Calendar Queue scheduler

Differences:

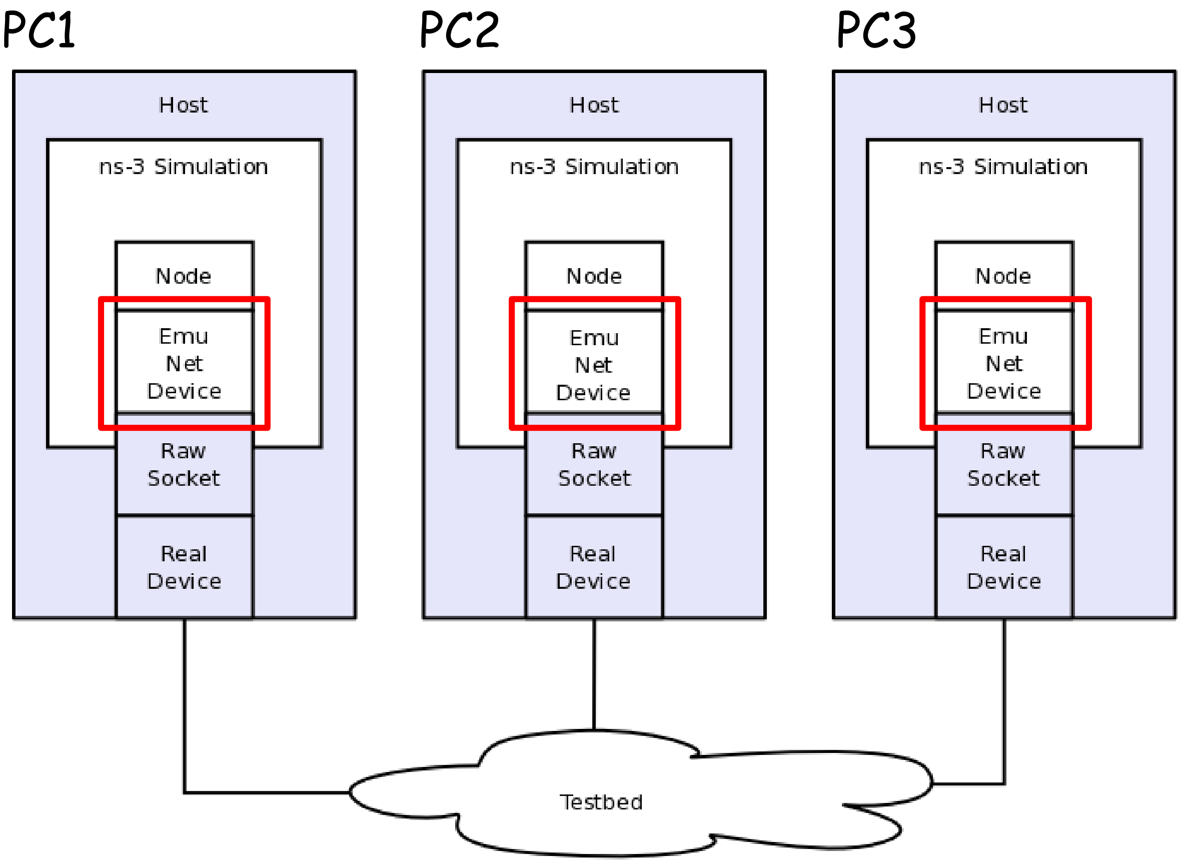
- Python scripting (or C++ programs) replaces OTcl
- most of the core was rewritten
- new animators, configuration tools, etc. are in work
- ns-2 is no longer actively maintained/supported

http://www.nsnam.org/docs/ns-3-overview.ppt

emulation

Emu NetDevice:

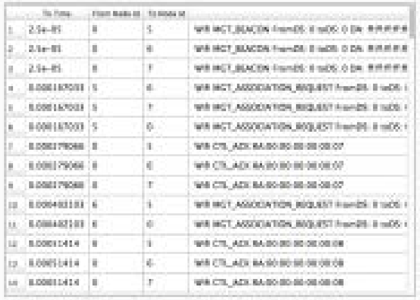
testbed usage



NetAnim

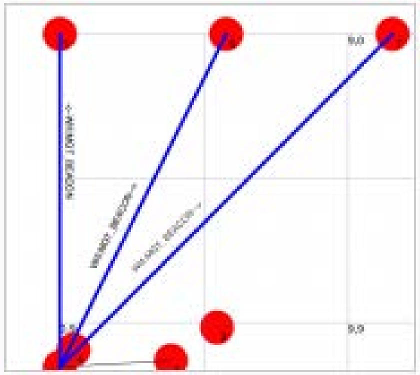
Visualization tool

similar to nam in ns-2

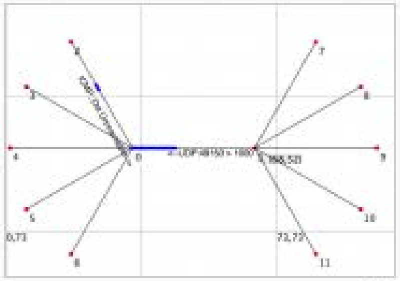


	Tx Time	From Node Id	To Node Id	
1	2.5e-05	0	5	WR MGT_BEACON fromDS: 0 toDS: 0 DA: ######
2	2.5e-05	0	6	WR MGT_BEACON fromDS: 0 toDS: 0 DA: ######
3	2.5e-05	0	7	WR MGT_BEACON fromDS: 0 toDS: 0 DA: ######
4	0.000167033	5	6	WR MGT_ASSOCIATION_REQUEST fromDS: 0 toDS:
5	0.000167033	5	7	WR MGT_ASSOCIATION_REQUEST fromDS: 0 toDS:
6	0.000167033	5	0	WR MGT_ASSOCIATION_REQUEST fromDS: 0 toDS:
7	0.000179066	0	5	WR CTL_ACK RA:00:00:00:00:00:07
8	0.000179066	0	6	WR CTL_ACK RA:00:00:00:00:00:07
9	0.000179066	0	7	WR CTL_ACK RA:00:00:00:00:00:07
10	0.000402103	6	5	WR MGT_ASSOCIATION_REQUEST fromDS: 0 toDS:
11	0.000402103	6	0	WR MGT_ASSOCIATION_REQUEST fromDS: 0 toDS:
12	0.00051414	0	5	WR CTL_ACK RA:00:00:00:00:00:08
13	0.00051414	0	6	WR CTL_ACK RA:00:00:00:00:00:08
14	0.00051414	0	7	WR CTL_ACK RA:00:00:00:00:00:08

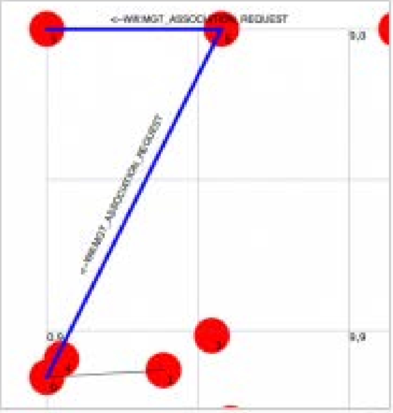
Packet Statistics



Wifi Beacon

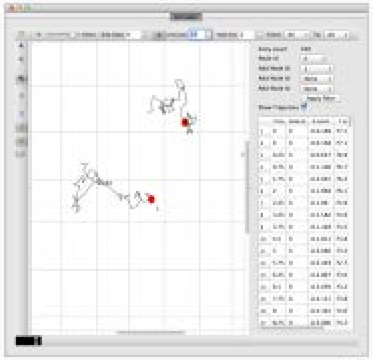


Dumbbell

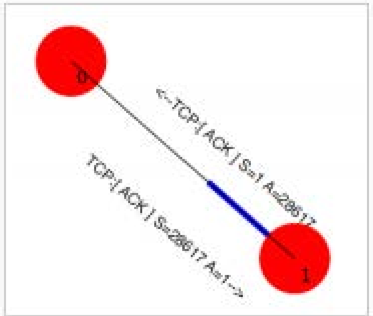


<--Wifi:MGT_ASSOCIATION_REQUEST
<--Wifi:MGT_ASSOCIATION_REQUEST
9,0
0,0
9,9

Wifi Assoc



Node Trajectory



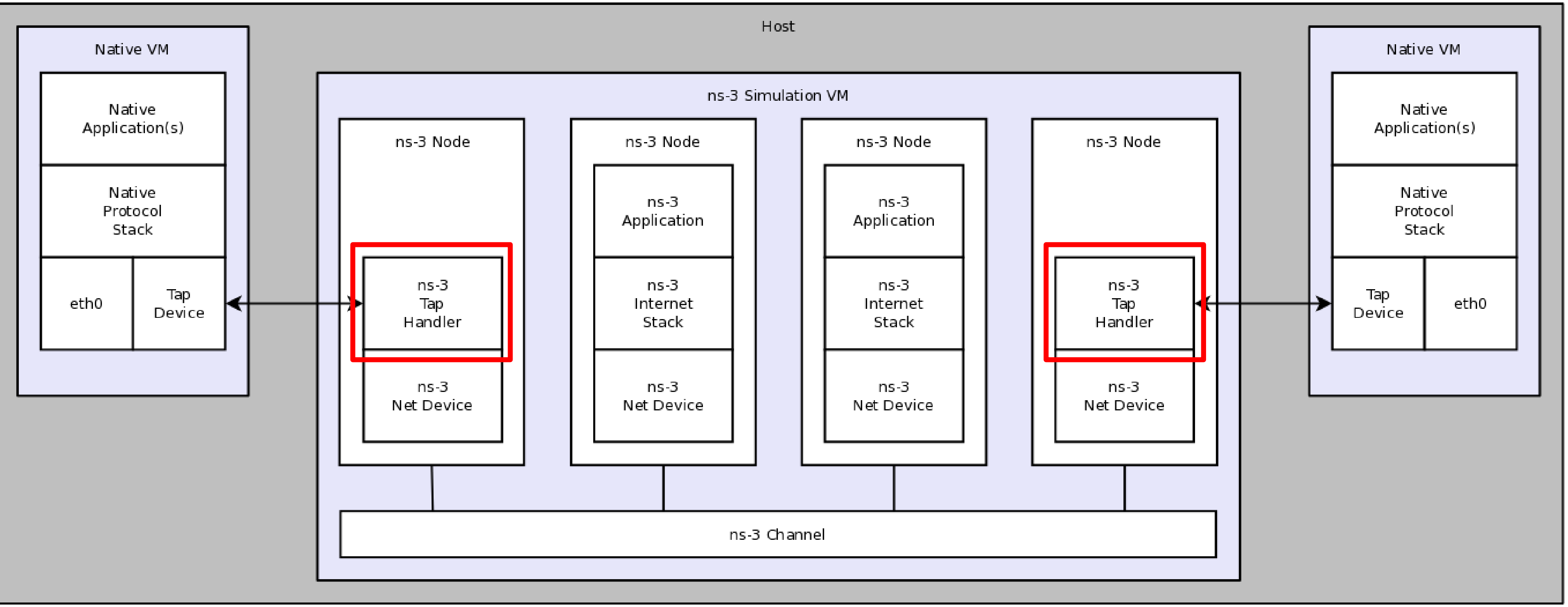
TCP flags

http://www.nsnam.org/docs/ns-3-overview.ppt

emulation

Tap NetDevice:

TapBridge Model: incorporation of native VMs into simulation



ns-3 Model Library

DCE (Direct Code Execution)

Simulation with Linux kernel implemented network protocol

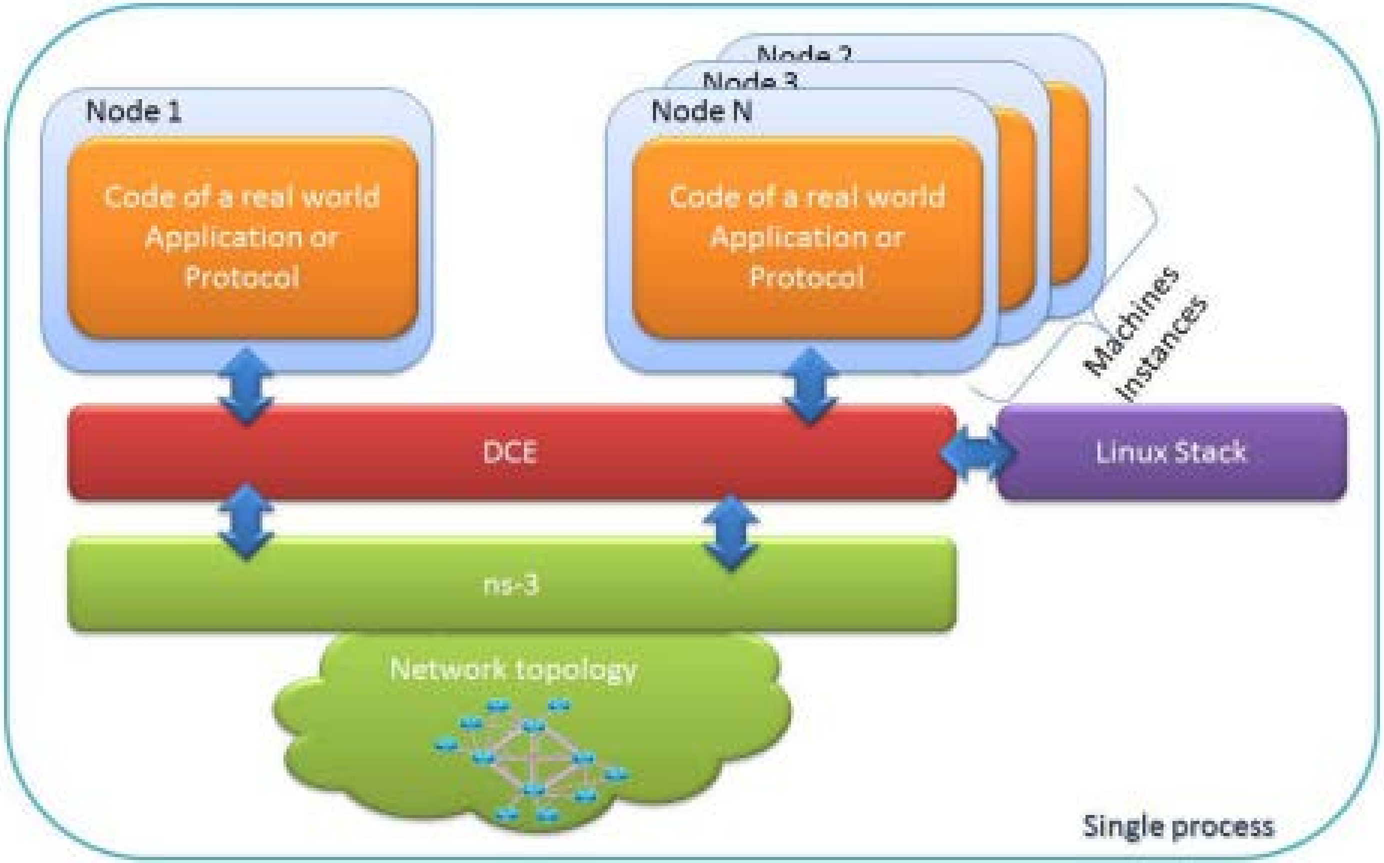
- IPv4/IPv6
- TCP/UDP/DCCP
- running with POSIX socket applications and ns-3 socket applications
- configuration via sysctl-like interface
- multiple nodes debugging with single gdb interface

similar to ns2 TCP Linux

https://www.nsnam.org/docs/dce/manual/singlehtml/index.html

DCE (Direct Code Execution)

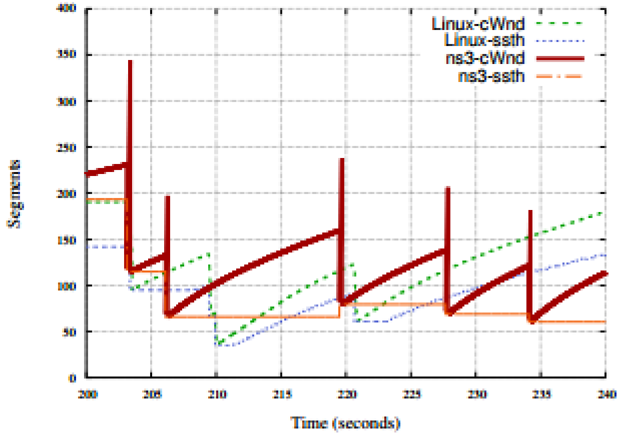
DCE enables using native Linux codes in ns-3 simulation



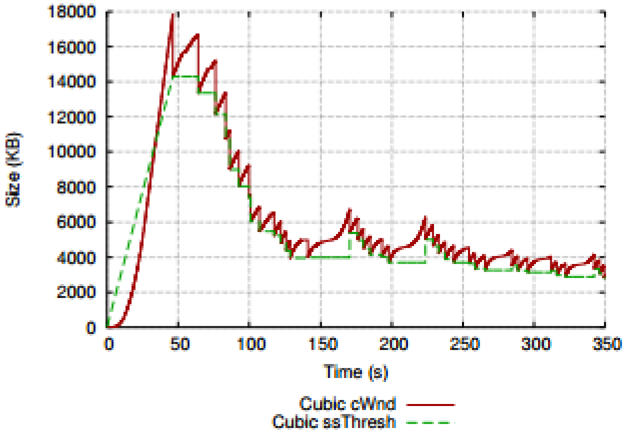
https://hal.archives-ouvertes.fr/file/index/docid/927519/filename/paper_final.pdf

TCP in ns3

- New Reno
- High Speed
- Hybla
- Westwood
- Vegas
- Scalable
- Veno
- Bic/Cubic
- YeAH
- Illinois
- H-TCP



New Reno



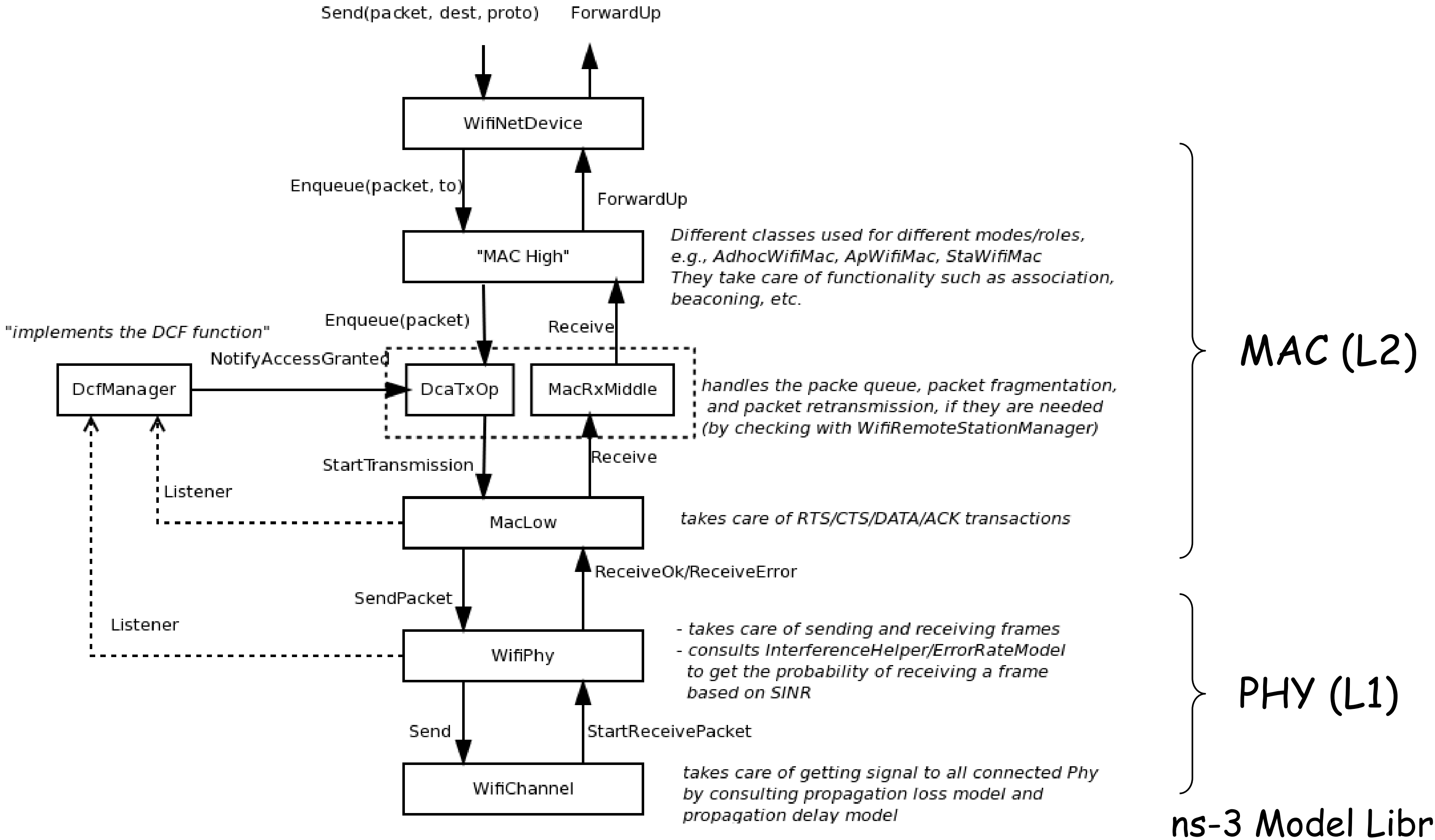
Cubic

http://dl.acm.org/citation.cfm?id=2756518

WiFi

WiFi NetDevice:

support 802.11a/b/e/g/n/ac



Send(packet, dest, proto)

ForwardUp

WifiNetDevice

Enqueue(packet, to)

ForwardUp

"MAC High"

Different classes used for different modes/roles, e.g., AdhocWifiMac, ApWifiMac, StaWifiMac They take care of functionality such as association, beaconing, etc.

"implements the DCF function"

Enqueue(packet)

Receive

NotifyAccessGranted

DcfManager

DcaTxOp

MacRxMiddle

handles the packe queue, packet fragmentation, and packet retransmission, if they are needed (by checking with WifiRemoteStationManager)

StartTransmission

Receive

Listener

MacLow

takes care of RTS/CTS/DATA/ACK transactions

ReceiveOk/ReceiveError

SendPacket

Listener

WifiPhy

- takes care of sending and receiving frames

- consults InterferenceHelper/ErrorRateModel to get the probability of receiving a frame based on SINR

Send

StartReceivePacket

WifiChannel

takes care of getting signal to all connected Phy by consulting propagation loss model and propagation delay model

MAC (L2)

PHY (L1)

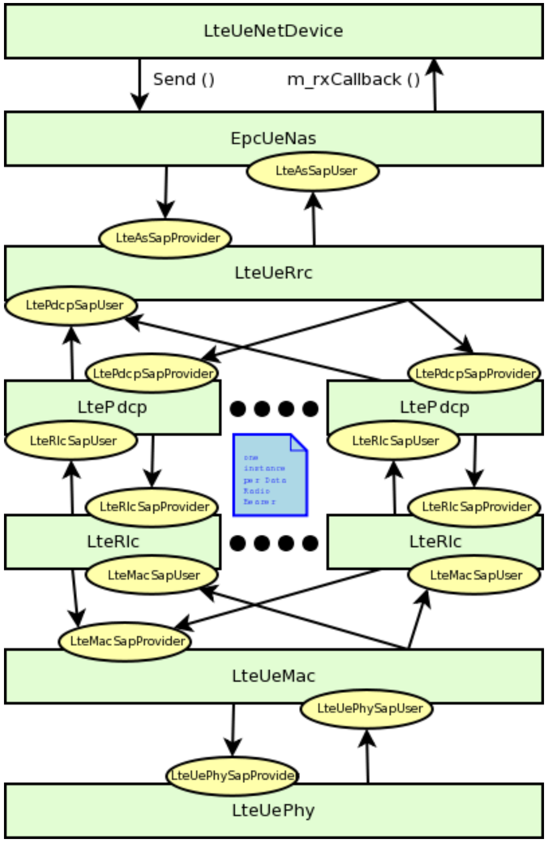
ns-3 Model Library

LTE

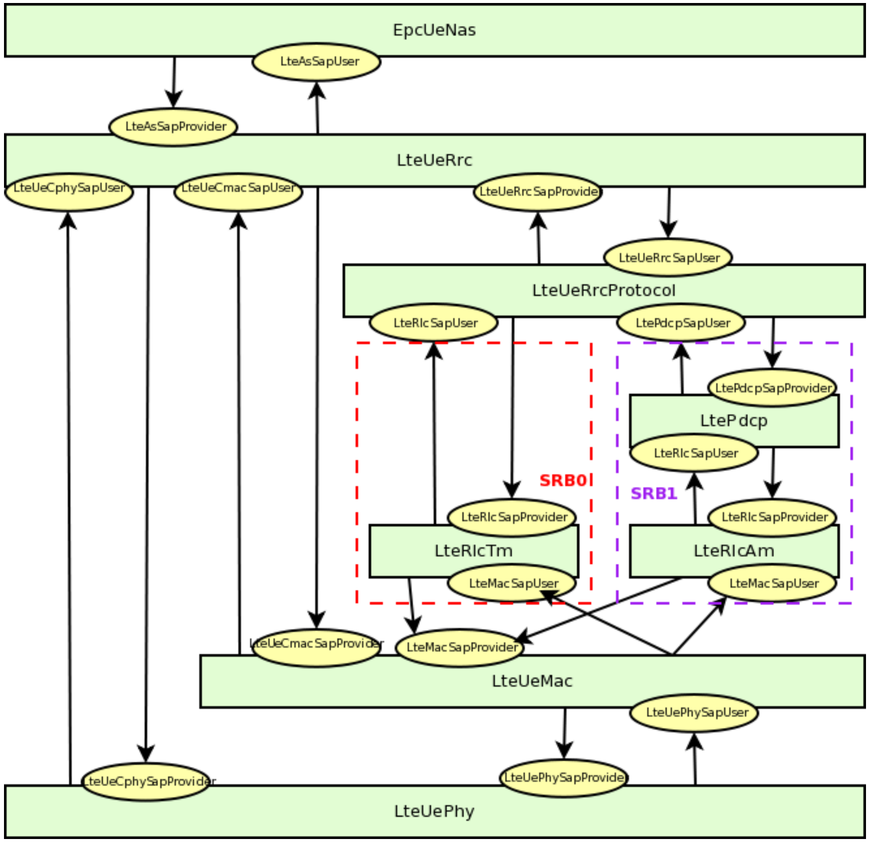
LteUe NetDevice:

UE: User Equipment

data plane



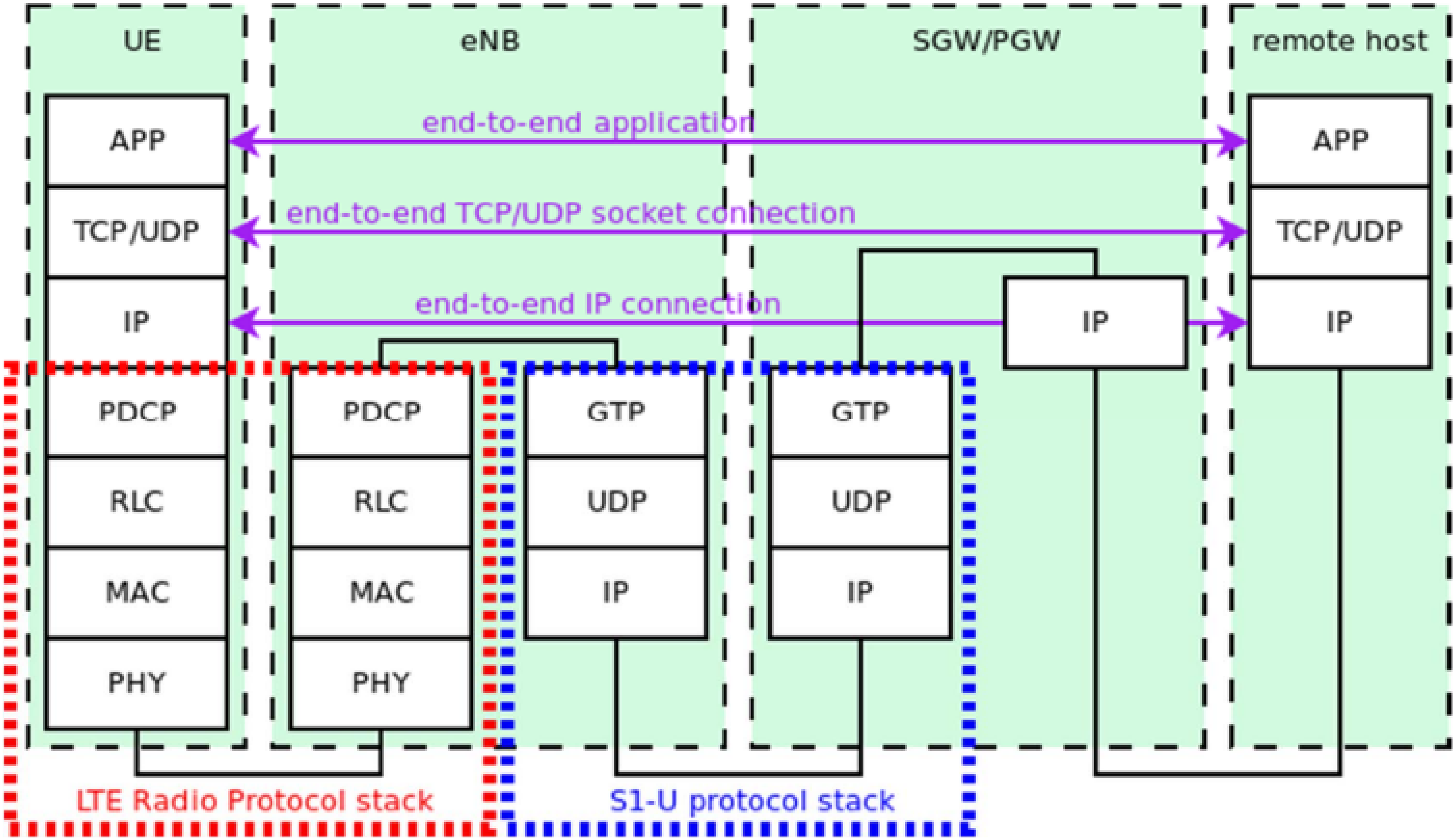
control plane



ns-3 Model Library, LENA project

LTE

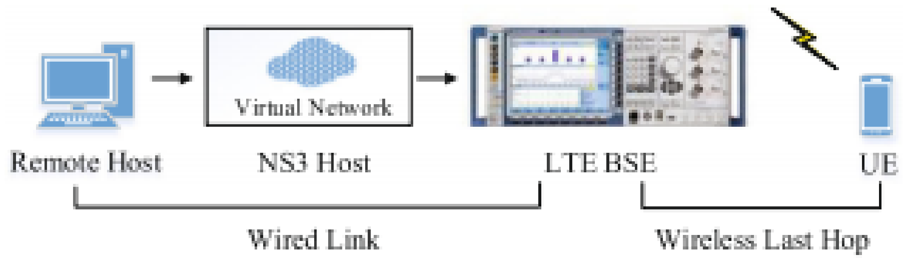
End-to-end data plane protocol stack



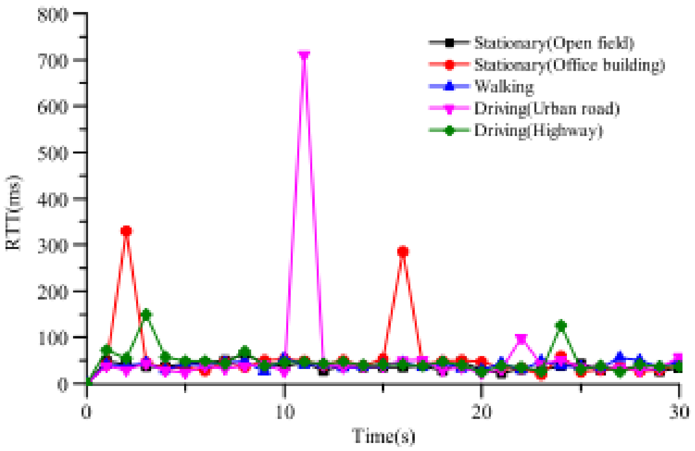
ns-3 Model Library, LENA project

Example of ns-3 experiment

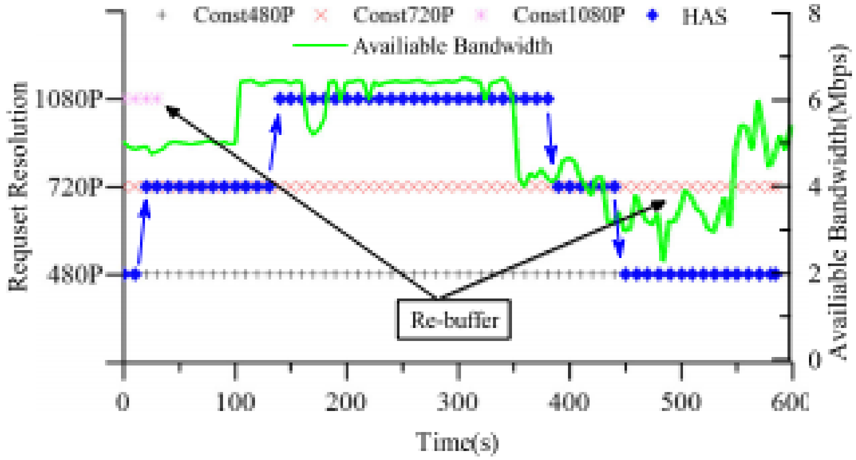
NS-3 emulation mode



BSE: Base Station Emulator



RTT measurement



HTTP adaptive streaming (HAS)

H.Du et al: "LTE-EMU: A High Fidelity LTE Cellar Network Testbed for Mobile Video Streaming," May.2017

Resources

Web site:

http://www.nsnam.org

Tutorial:

https://www.nsnam.org/documentation/

Code server:

http://code.nsnam.org

Wiki:

https://www.nsnam.org/wiki/index.php/Main_Page