



Designing Network Protocols and Applications

Approaches to design, engineer and validate network protocols

Florian Westphal

fw@strlen.de

Hagen Paul Pfeifer

hagen.pfeifer@protocollabs.de

14. April 2009

Topic

In the 25+ years the TCP/IP protocol family has been deployed on the internet, it has often faced new problems and challenges. Even today, new extensions such as ECN and TCP's user timeout option, are being added to the core protocols.

Newer network protocols, such as DCCP, are being tuned to better adapt to real world behaviour (e.g. reaction to packet loss). New physical networks, such as wireless meshes, are deployed.

But how are new protocols, extensions to existing protocols, or even the performance of protocols tested? How can we validate that some change works as expected?

This talk aims to cover a wide range of approaches to ease the design and validation of networking protocols before wide spread deployment.

Agenda

- ▶ Network simulation
 - ns-2
 - ns-3
- ▶ Supplementary approaches and tools
 - Emulation, ...
 - Analysis tools

Network Simulation

- ▶ It is often not quite possible to verify a system adequate
 - Huge network topologies, heterogeneous infrastructure, . . .
- ▶ One solution: simulate the behavior of the system
 - Simulation represent a constructed and abstract model of the network
 - Leave out components of minor interest (e.g. a too accurate PHY model or application model, et. cetera) — or shrink to a bare (absolutly required) minimum
- ▶ Measurements and experiments are limited in their usage
 - Only existing infrastructure can be tested/analysed
 - Scales poorly for size and effort
 - No complete control and insights (often a extensive, intrinsic knowledge is the purpose)
- ▶ Simulation is the key component in network research
- ▶ Last but not least: network simulators helps enormously to develop intuition!

Network Simulation

► Simulation layers:

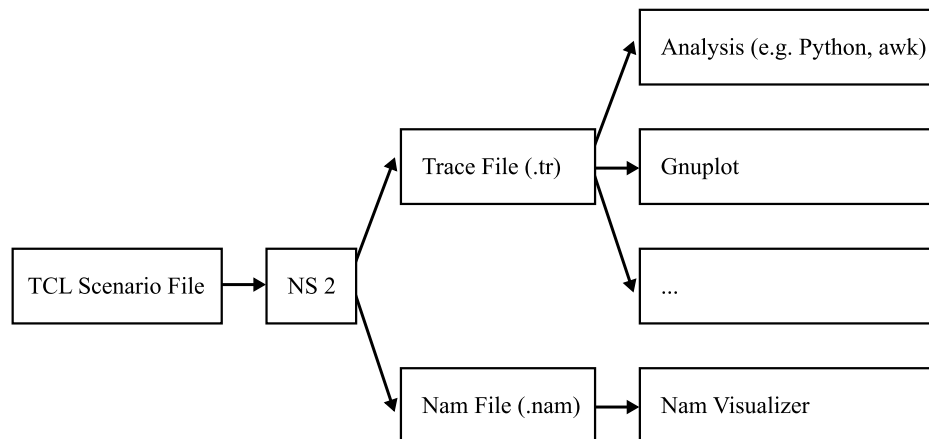
- Application layers (FTP, Telnet, CBR, VBR, P2P, ...)
- Transport layers (TCP, UDP, SCTP, ...)
- Network layers (IP, IPv6, Routing Protocols (OSPF, RIP, OLSR, AODV, ...), static routing)
- MAC layers (802.3, 802.11, TDMA, ...)
- Physical layers (QPSK, QAM n , ...)
- Channel characteristics (Friis, Two Ray Ground, Nakagami, ...)
- Node mobility (Random Walk Mobility Model, ...)
- ...

Network Simulation

- ▶ Discrete event driven simulator – Functioning
 1. Activities are translated into events
 2. Events are queued and processed in the order of their scheduled occurrences
 3. Time progresses as the events are processed
- ▶ Common Network Simulators
 - OMNeT++ (OPNET)
 - ns-2, ns-3
 - j-sim
 - SSFNet
 - GloMoSim
 - QualNet
 - GTNeTS

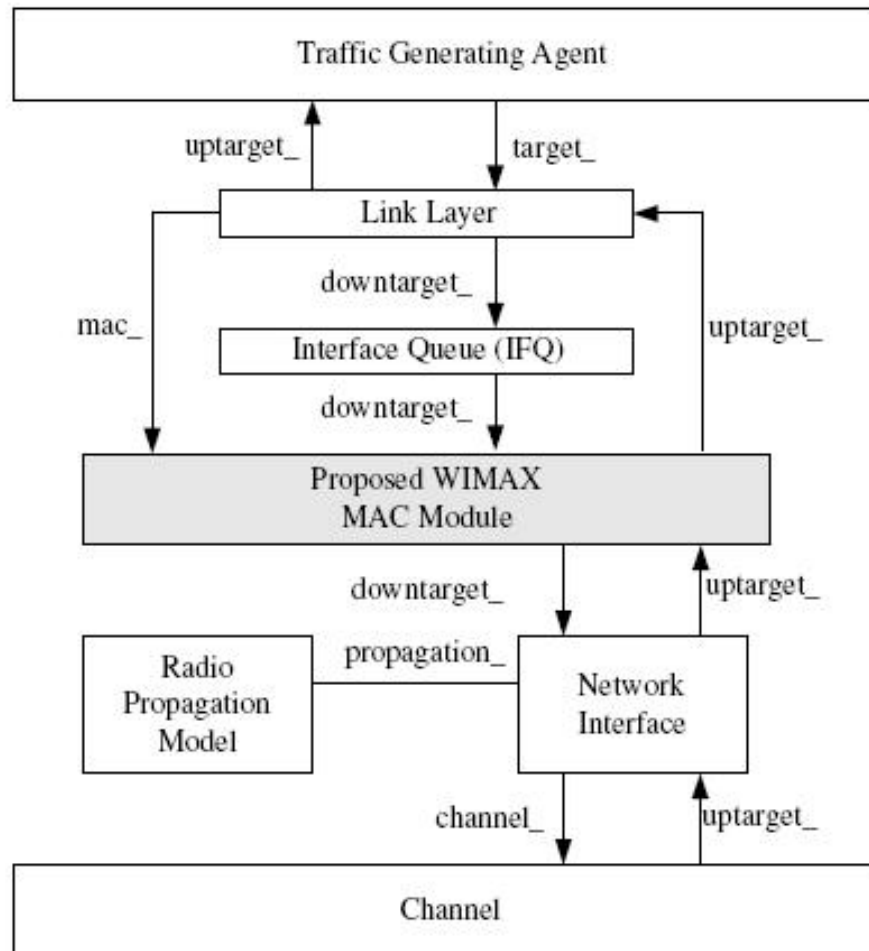
ns-2

- ▶ Discret network event simulator
- ▶ $\geq 50\%$ of ACM and IEEE papers cite ns2
- ▶ Consists of a C++ core and an OTcl interpreter “frontend”
- ▶ Network topology is set up using OTcl script
- ▶ Workflow:



ns-2 Architecture

- Architecture (WIMAX example):



ns-2

- ▶ But, ...
 - Maintaining stalled at a absolute minimum
 - Development process stalled already
 - Bad source code basis (C++)
- ▶ What is the new shining star at the sky?

ns-3

- ▶ Successor to ns-2; under development
- ▶ ns-3 is a library written in C++, there is no "ns3" executable
- ▶ Simulation/network topology is set up in C++ or python
- ▶ Workflow: programmatic description of network
 - Create node objects
 - Create network topology
 - Assign addresses, routes, etc.
- ▶ Simulation is run by the ns-3 scheduler, time moves from event to event
- ▶ Workflow:
python Simulation Script / C++ program → pcap file → analysis (eg. `wireshark`, `tcptrace`, ...)

Outputs: pcap traces (one per interface); customizable trace namespace to get particular events (eg. IPv4 tx on interface i_1)

ns-3 Features

▶ Emutap:

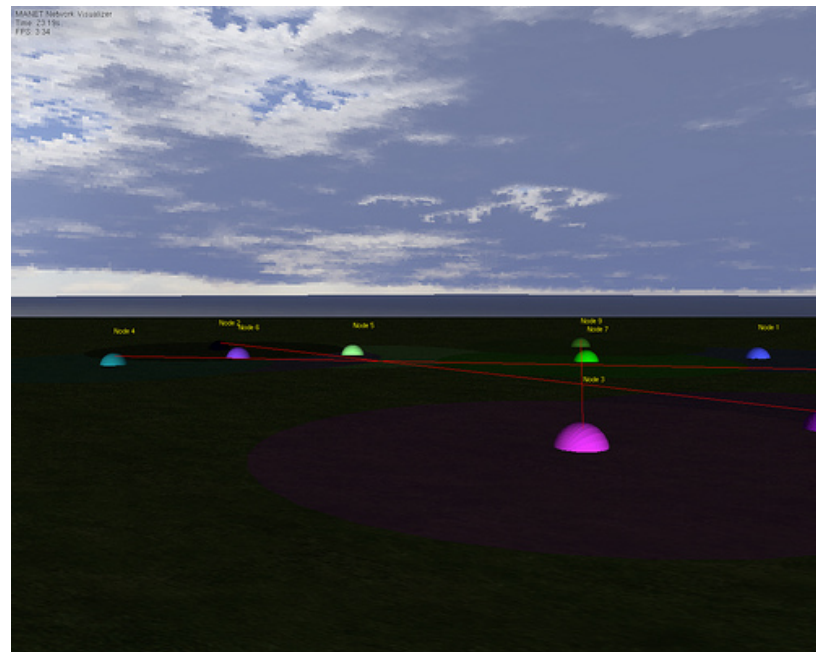
- Connect real testbed and simulator:
send & receive packets from network via tun/tap interface

▶ ns-3-simu:

- Custom ELF loader to run unmodified Linux x86 ELF binaries within ns-3
- `ping(8)` works ;-)

▶ Tracing Framework

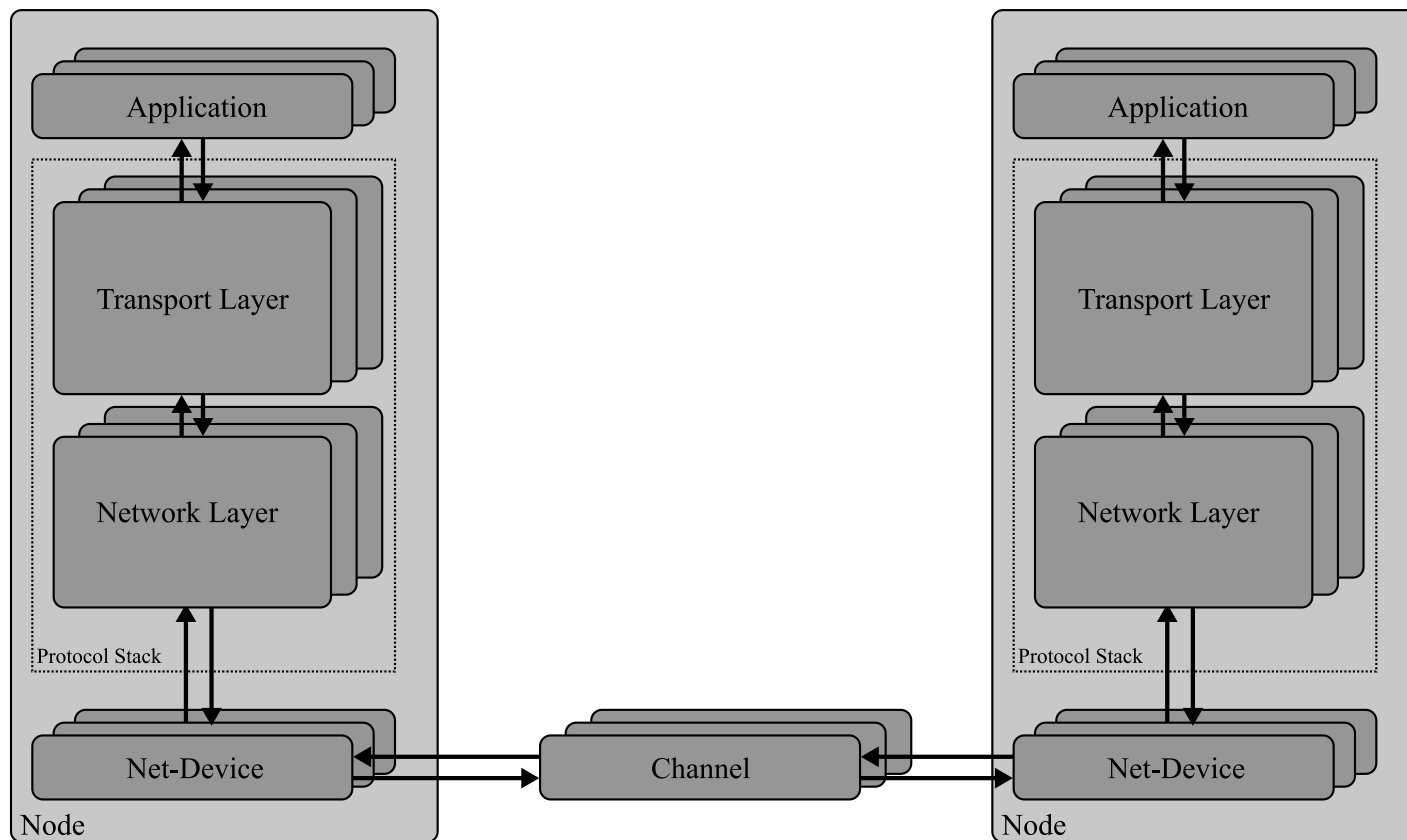
▶ Visualization



ns-3 Nomenclature/Concepts

- ▶ A simulated network has **Nodes** (“computers”)
- ▶ Each node uses a particular protocol stack, e.g. TCP/IP
- ▶ Node can run applications that talk to network via socket-like interface
- ▶ Nodes have one or more **NetDevices** (“network interfaces”)
- ▶ network interfaces are connected to particular **Channels**, e.g. PPP, CSMA, ...
- ▶ Data is exchanged using **Packets**
 - Additional information can be added via tags

ns-3 Nomenclature/Concepts



Network Simulation Cradle – NSC

- ▶ Developed by Sam Jansen at WAND, 1st release 2005
- ▶ Wrapper/compat layer to run various network stacks in user space
→ creates `liblinux2.6.26.so`, etc.
- ▶ Not a network simulator itself, can be interfaced with ns-2, ns-3 and others
- ▶ Allows use of FreeBSD 5.3, OpenBSD 3.5 and Linux 2.6.26 network stacks in ns-2 and ns-3 simulations
- ▶ sysctl support
`("stack->sysctl_set("net.ipv4.tcp_congestion_control", "hybla"))"`

TCP Stack ↔ NSC ↔ Network Simulator

Network Simulation Cradle – NSC

Why use real network stacks?

- ▶ they behave different than those implemented in ns-2, ns-3, etc.
- ▶ Can be used to find problems when different stacks interact (eg. broken Timestamps in SYN segment)
- ▶ Can be used to validate prototype implementations before running them in kernel mode
- ▶ You can run the simulator in a debugger
- ▶ Very useful for learning: e.g. set breakpoints with gdb, single-step through IP stack, etc.
- ▶ TODO/Wishlist:
 - Full tcp/ip routing capabilities
 - IPv6
 - AQM
 - ...

NSC Implementation

- ▶ The cradle provides the necessary infrastructure for each network stack
- ▶ Fake network driver (`"nsc0"`)
- ▶ Lots of support code (e.g. Linux):
 - `copy_from_user` → `memcpy`, ...
 - Kernel start up routine (`"do_initcalls"`)
- ▶ Cradle also defines API to map syscalls and set up the stack
- ▶ Defines timer interrupt method that is called by the simulator periodically

Another Problem: A stack is usually only for a single host

- ▶ need separate stacks for each node in the simulation
- ▶ NSC solution: (Mostly) Automated source code transformation using the `nsc globalizer`:

```
"long global_var → long global_var[NUM_STACKS]"
```

```
"global_var = x → global_var[get_stack_id()] = x"
```


TCP related tool

▶ TcpProbe

- Kernel module to trace TCP internals

▶ US Tools

- Netperf, Iperf, ttcp, netpipe, netsend, ...

▶ tcptrace

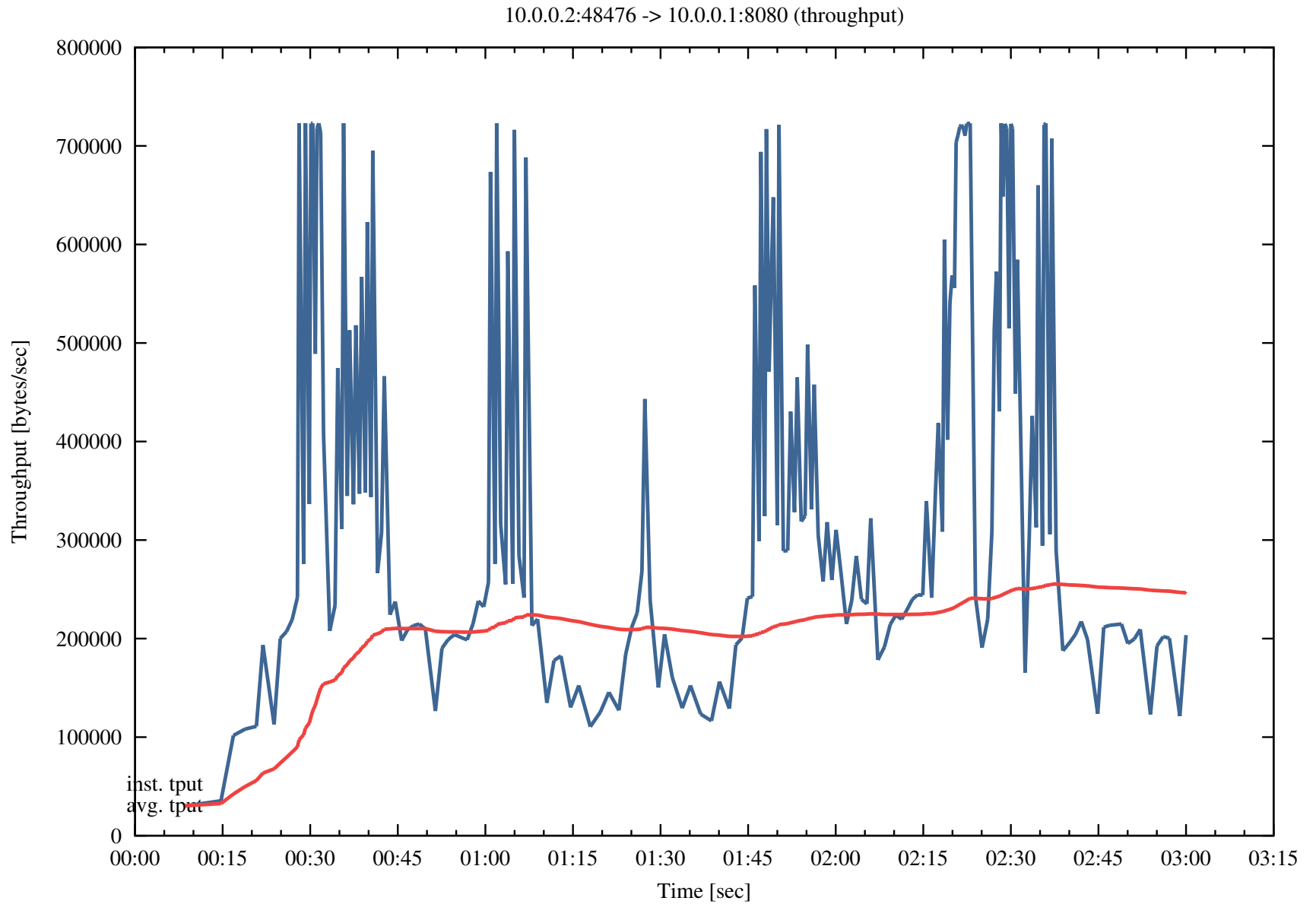
- TCP connection analysis tool
- summarizes content of pcap files
- rtt statistics, throughput, ausstehende daten, time sequence...
- creates output data for xplot (`gnuplot` users convert format via `xpl2gpl`)

▶ HTTP

- Generate representative HTTP workloads for network and server performance evaluation
- HTTP state machine

- 
- Example: Monkey See, Monkey Do: A Tool for TCP Tracing and Replaying

TCP Trace



Pktgen

- ▶ Kernel tool to generate packets at a high packet rate

- ▶ Implemented through kernel threads

- ▶ Controlled via proc file system

```
echo add_device eth0:0 > /proc/net/pktgen/kpktgend_0
```

```
echo dst 10.0.0.1 > /proc/net/pktgen/eth0:0
```

- ▶ Really aggressive in using hardware resources:

- Avoid in kernel processing overhead

- One cloned `sk_buff`

- No userspace \leftrightarrow kernel path

Netem

- ▶ Linux Kernel Network Emulator
- ▶ Common uses:
 - Emulate behavior of wide area networks
- ▶ Controlled by `tc(8)` (`iproute2`)
- ▶ Tunable knobs:
 - Packet delay
 - Packet loss
 - Packet duplication
 - Packet corruption
 - Packet re-ordering
- ▶ `tc qdisc add dev eth0 root netem delay 100ms`
- ▶ `tc qdisc change dev eth0 root netem delay 100ms 10ms`

▶ `tc qdisc change dev eth0 root netem delay 100ms 10ms 25%`

▶ Examples:


- How does TCP react by a packet loss factor of 10%?
- Emulate a 56k modem line?
- What is the sensible impact of a 300ms delay for VoIP?

▶ Similar: NIST Net

▶ FreeBSD pendant: dummynet

Literatur

- [1] John Bellardo and Stefan Savage. Measuring packet reordering. In *In ACM SIGCOMM Internet Measurement Workshop*, pages 97–105, 2002.
- [2] Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, John Heidemann, Ahmed Helmy, Polly Huang, Steven McCanne, Kannan Varadhan, Ya Xu, and Haobo Yu. Advances in network simulation. *Computer*, 33(5):59–67, 2000.
- [3] Kenneth L. Calvert, Matthew B. Doar, Ascom Nexion, Ellen W. Zegura, Georgia Tech, and Georgia Tech. Modeling internet topology. *IEEE Communications Magazine*, 35:160–163, 1997.
- [4] Jin Cao, William S. Cleveland, Yuan Gao, Kevin Jeffay, F. Donelson Smith, and Michele Weigle. Stochastic models for generating synthetic http source traffic. In *In IEEE INFOCOMM*, pages 1546–1557, 2004.
- [5] Yu chung Cheng, Urs Hölzle, Neal Cardwell, Stefan Savage, and Geoffrey M. Voelker. Monkey see, monkey do: A tool for tcp tracing and replaying. In *In USENIX Annual Technical Conference*, pages 87–98, 2004.
- [6] K. G. Coffman, K. G. Coffman, A. M. Odlyzko, and A. M. Odlyzko. Growth of the internet. In *Utility, Utilization, and Quality of Service, Tech. Rep. 99-08, DIMACS*, pages 17–56. Academic Press, 2001.
- [7] Sally Floyd and Eddie Kohler. Internet research needs better models. In *in HotNets-I*, 2002.
- [8] Sally Floyd and Vern Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9:392–403, 2001.
- [9] Andrei Gurtov. Responding to spurious timeouts in tcp. In *In Proc. of IEEE INFOCOM 2003*, 2003.
- [10] Michael Liljenstam, David M. Nicol, Vincent H. Berk, and Robert S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *in Proceedings of the 2003 ACM workshop on Rapid Malcode*, pages 24–33. ACM Press, 2003.
- [11] Jitendra Padhye and Sally Floyd. Identifying the tcp behavior of web servers. In *In ACM SIGCOMM*, 2001.
- [12] V. Paxson, M. Allman, S. Dawson, W. Fenner, J. Griner, I. Heavens, K. Lahey, J. Semke, and B. Volz. Known tcp implementation problems. United States, 1999. RFC Editor.

- 
- [13] Vern Paxson. Measurements and analysis of end-to-end internet dynamics. Technical report, 1997.
 - [14] Vern Paxson, J. Kurose, C. Partridge, and E. W. Zegura. End-to-end routing behavior in the internet. In *IEEE/ACM Transactions on Networking*, pages 601–615, 1996.
 - [15] Georgos Siganos, Michalis Faloutsos, and Christos Faloutsos. The evolution of the internet: topology and routing.
 - [16] Walter Willinger and Vern Paxson. Where mathematics meets the internet. *Notices of the American Mathematical Society*, 45:961–970, 1998.

Contact

▶ Hagen Paul Pfeifer <hagen@jauu.net>

- Key Id: 0x98350C22

- Key Fingerprint: 490F 557B 6C48 6D7E 5706 2EA2 4A22 8D45 9835 0C22

▶ Florian Westphal <fw@strlen.de>

- Key Id: 0xF260502D

- Key Fingerprint: 1C81 1AD5 EA8F 3047 7555 E8EE 5E2F DA6C F260 502D

netkit

- ▶ Emulate a virtual network via UML (User Mode Linux)
- ▶ Application
 - setup of smaller network infrastructure
 1. Routers (via quagga)
 2. Switches (
 3. Computers (linux application omnium gatherum)
- ▶ Usage
 - Specify network topology via configuration file (XML)
- ▶ Examples:
 - MPLS
 - RIP
 - OSPF
 - IS-IS

- BGB
- SNMP
- ▶ Similar projects
 - UMLMON
 - Emulab

kprobes

► global value

```
global netpstats
```

```
global cpustats
```

```
global init_time
```

```
probe begin{
```

```
printf("Press ^C to stop\n")
```

```
init_time = gettimeofday_ms()
```

```
}
```

```
probe module("tg3").function("tg3_poll"){
```

```
value = gettimeofday_ns()
```

```
cpustats <<< cpu()
```

```
}
```

```
probe module("tg3").function("tg3_poll").return{
```

```
diff = gettimeofday_ns() - value
```

```
netpstats <<< diff
```

```
}  
probe end {  
    print(@hist_linear(netpstats,0, 15000, 500))  
    print(@hist_log(cpustats))  
    printf("Total time: %d miliseconds\n", gettimeofday_ms() - init_time)  
}
```