Network Emulation

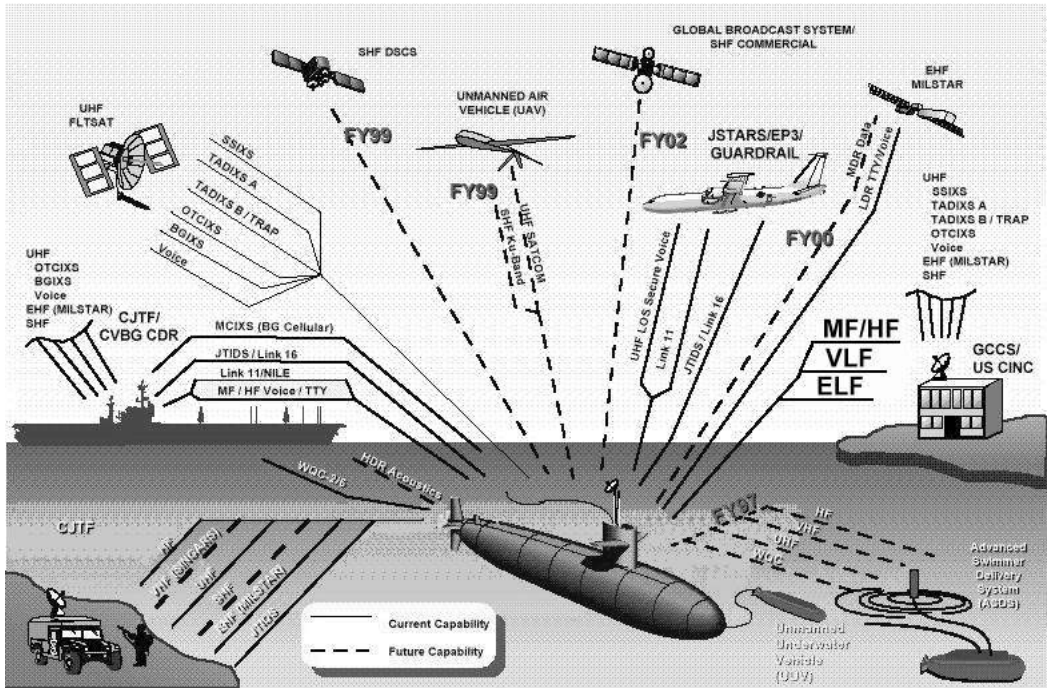
Overview, State of the Art and Current Development

Hagen Paul Pfeifer
hagen.pfeifer@protocollabs.com

ProtocolLabs
http://www.protocollabs.com

Introduction

- In the early days the network was the testbed
- But network protocols, applications and network characteristic are often too complex
- To analyze, validate or develop "something"[1] you have two choices:
  1. Network Simulation
  2. Network Emulation



[1] "something" can be a network protocol, a network application, a queue, a communication scenario and all kinds of tests where a network is involved

Network Emulation - Introduction II

- With emulation it is possible to emulate really realistic end-to-end scenarios[2]
  - It is more likely that you fail to characterize the network at that level than an emulator is able to emulate!
- You analyze/validate/develop within the target system
- Did you ever trust your simulation really?[3]
  - Random generator, TCP model, timing behavior, analysis scripts, traffic generator (Poison model), link layer collisions, ...
- Network simulators are qualified for prototyping, to get a bigger picture of basic functionality (e.g. simulate a large network topology with thousands of routers, rough TCP understanding)
- In the end: the choice of simulator vs. emulator depends on several factors. But if it is somehow possible to analyze/validate/develop your system using an emulator: take the chance!

[2] See netem paper from Stephen Hemminger In Linux Conf AU – 2005

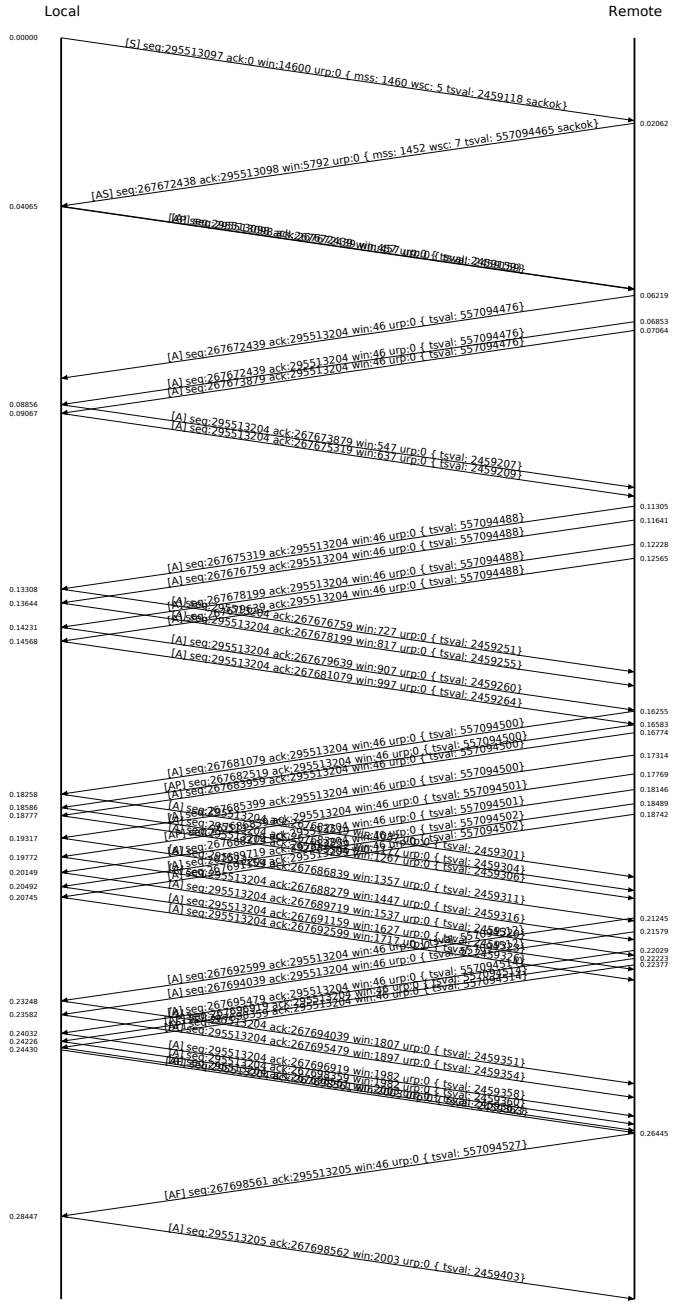
[3] Have you ever extended a simulator? Do you remember of that time? Do you remember the impact on a slightly changed variable for the system? A simulator is full of variables, are you sure *your* particular use-case is tested by the authors of the simulator? Did you remember of bugs in a real network system which arise because of the real-world complexity (e.g. Linux vs. Windows interoperability)? Are you sure you realize an error in the simulator if it will occur?

Network Emulation Overview

- NistNet
  - One of the first emulators
  - Linux kernel module (not developed anymore)
- Dummynet
  - FreeBSD, Mac OS X (and Linux)
- Netem
  - Enhancement of the Linux traffic control facilities
  - More features as dummynet
  - High Resolution Timer
  - Only Linux (kernel module)
- Emulate Larger Networks:
  - Virtualized networks with XEN, UML, KVM (see planetlab, emulab)

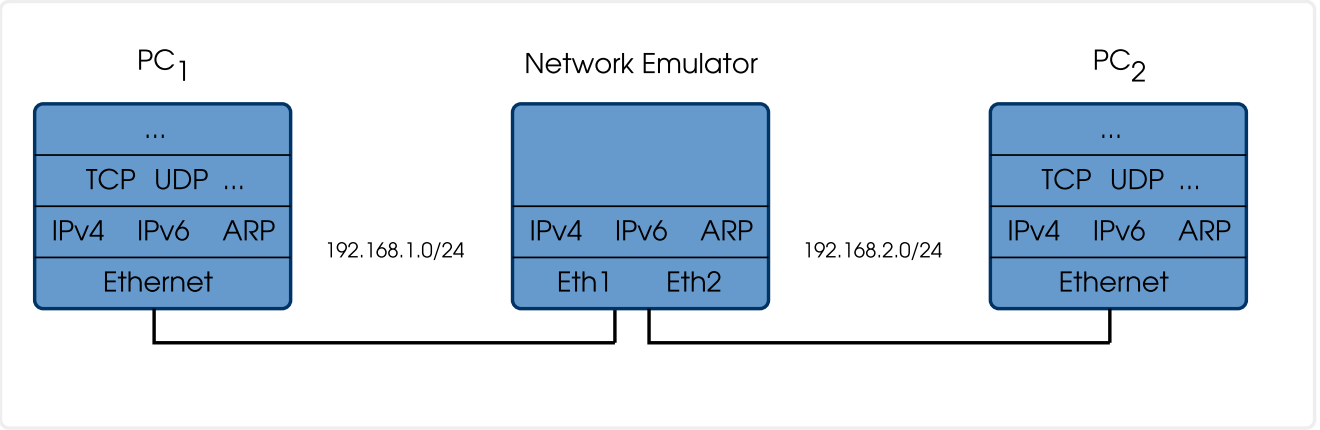
Network Emulation Possibilities

- Network delay (e.g. `500ms` for satellite links)
- Packet Corruption (wireless links, defect hardware)
- Packet Reordering (routing issues)
- Loss (congestion, lossy links)
- Duplication (defect hardware, routing anomalies)
- Link rate (e.g. throttle link to `100kbit/s`)



Example Setup

- Two network segments (IPv4, IPv6)[4]
- Separate emulation computer to reduce clock[5] issues
- Emulator act as an ordinary IP router

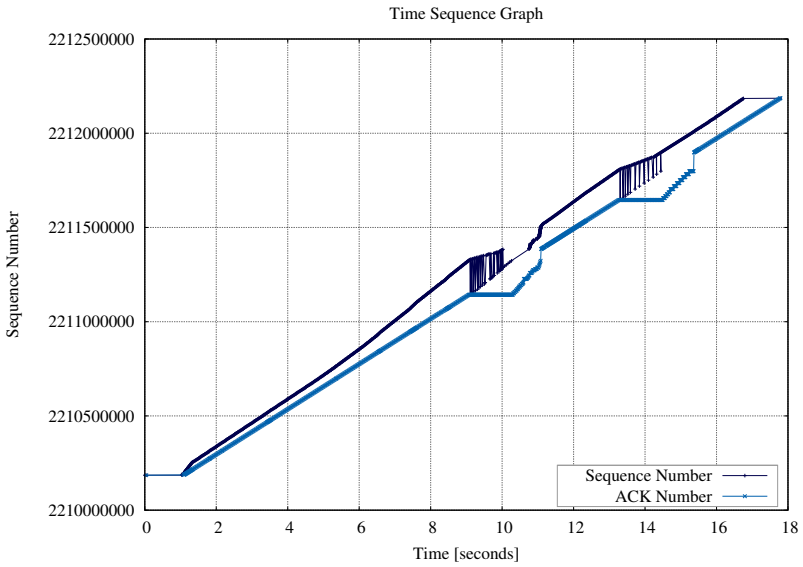


[4]VLAN separation possible

[5]OS clock granularity may an issue in some setups

Delay

- Delay with random jitter (and correlation)
- Example:
  - `tc qdisc add dev eth0 root netem delay 100ms 10ms 10%`
- Be warned:
  - Network adapters can also delay packet in their “queues” (ring-buffer)
  - Bounded to the kernel timing system. Depending on the architecture timer granularity, higher rates (e.g. `10mbit/s` and higher) tend to transmission bursts



Corruption

- Using Gilbert-Elliot/4-State-Markov model (based on work from S. Salsano, F. Ludovici, A. Ordine[6]

  `0000000000111110000000000111110000000000111110000000000011111` [7]

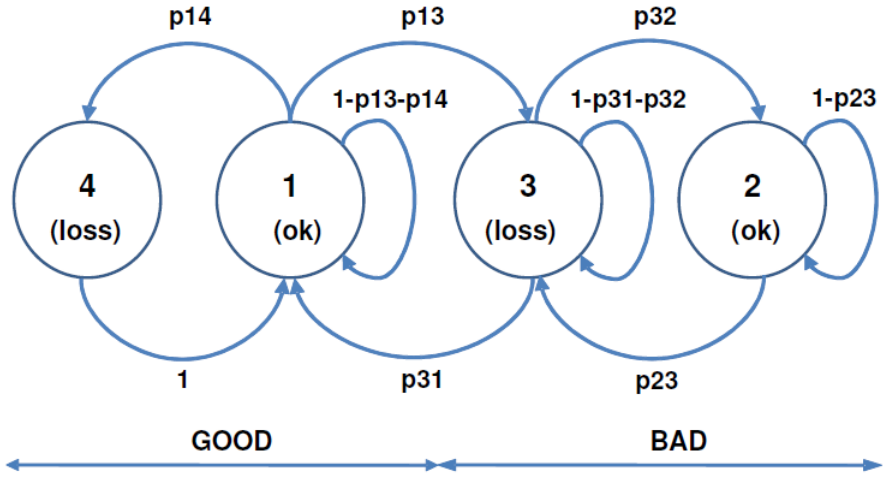
  - Certain percentage of constant noise AND contemporaneous emulate burst periods

    `0001000000111110010000000111110000010000111110000000010011111`

  - Also: possible to specify restfulness periods within a burst phase

    `0001000000111110010000000101110000010000110110000000010011111`

- Status: rfc patch



[6] http://netgroup.uniroma2.it/NetemCLG)

[7] 0 mean packet ok, 1 mean packet corrupted

Corruption II

- Bit corruption at random offset
- `start offset` (0 means start of packet)
- `end offset` (0 means end of packet, negative values to specify $tail - offset$)
- Several coding schemes use a stronger encoding for protocol header information
- Header checksum can be designed to protect only the header but not the payload
  - This can be of interrest for codec test: UDP/RTP/Voice[8] (UDPLite)
- Work in progress: hardware checksum features issues

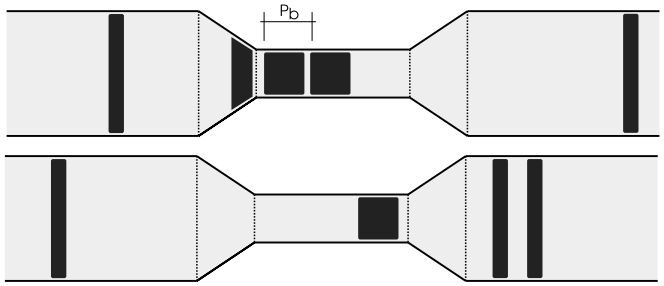
[8] cp. IPv6 and UDP header checksum requirements

Loss

- `loss random PERCENT [ CORRELATION ]`
- 1 of 1000: `tc qdisc change dev eth0 root netem loss 0.1%`
- Correlation:
  - `tc qdisc change dev eth0 root netem loss 0.1% 25%`
  - 0.1% packets to be lost, each successive probability depends by a quarter to the last one
  - $Prob_n = 0.25 * Prob_{n-1} + 0.75 * rand()$

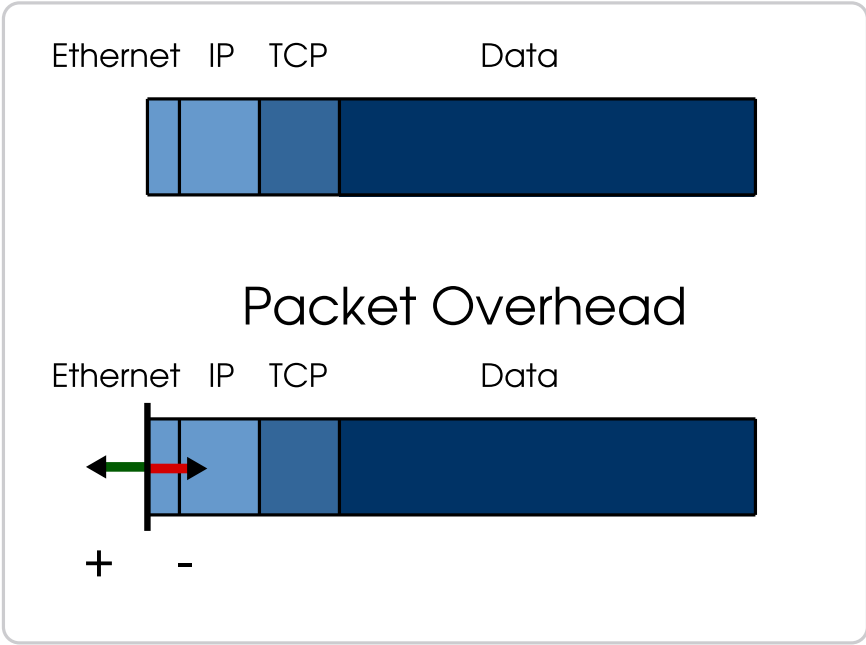
Rate Limiting

- Rate extension (Linux kernel version: 3.3)
- Up to now Token Bucket Filter (tbf) was used to shape packets (e.g. limit to 10kbit/s)
- But TBF has some fundamental flaws which cannot be fixed!
- Netem rate extension (commit 7bc0f28c7a0c)
  - `tc qdisc add dev eth0 root netem rate 10kbit`



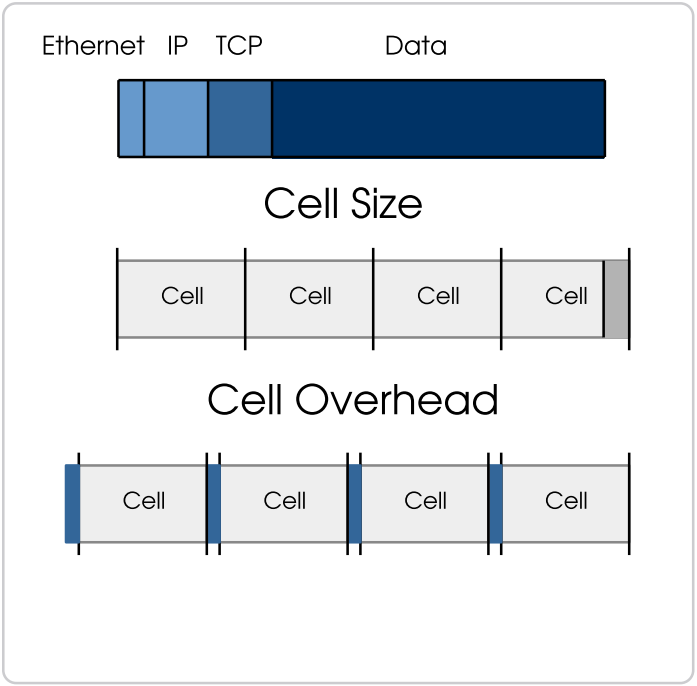
Packet Overhead

- Per packet
- Can be positive $(len(packet) + nbyte)$ or negative $(len(packet) - nbyte)$
- Can be used to simulate
  - IP/TCP Compression Schemes (e.g. ROHC)
  - Link Layer Encryption Overhead
  - Link Layer Header Overhead



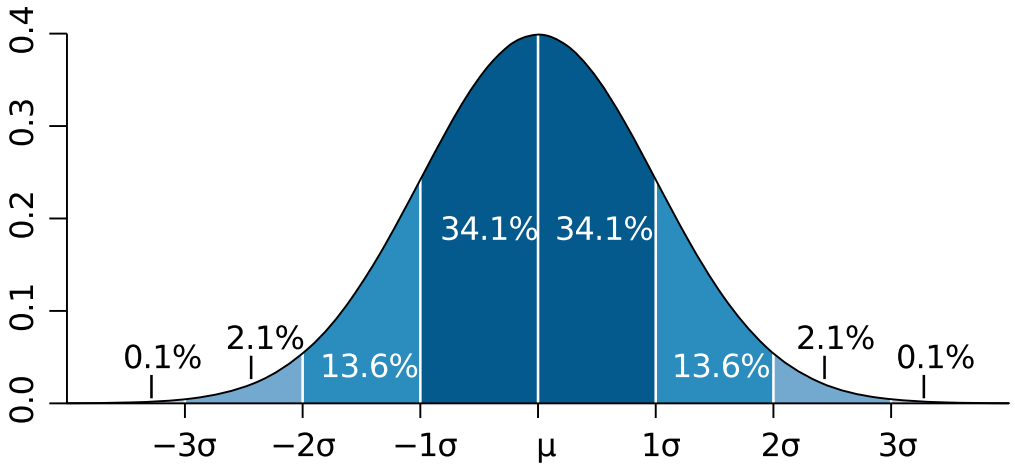
Cell Overhead

- Used to simulate link layer schemes which operates on cells (e.g. ATM, Link Layer Fragmentation, ...)
- Cellsize to specify the minimal/maximal cellsize
  - Packet must be split into proper cell size, last chunk cell size overhead is "lost"
  - Cellsize overhead to simulate cell packet header
- ATM:
  - Cell payload size: 47 byte
  - Cell header size: 5 byte



Distribution Tables

- Delay - based on NistNet
- Implemented by using Distribution Tables
- Following tables are shipped with `tc`
  - Normal distribution
  - Pareto distribution
  - Paretonormal distribution
- Tool provided to build generate own tables



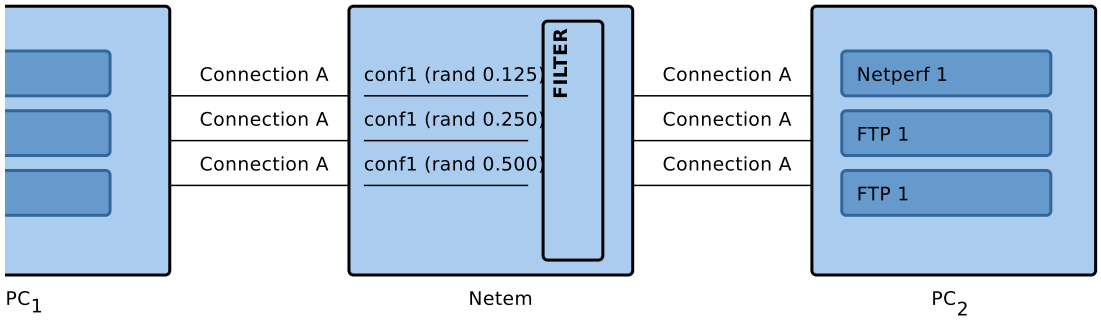
Queue Filter

- `tc filter add dev eth0 protocol ip par-`

  `ent 1: prio 1 u32 match ip src 1.2.3.0/24 flowid 1:10`

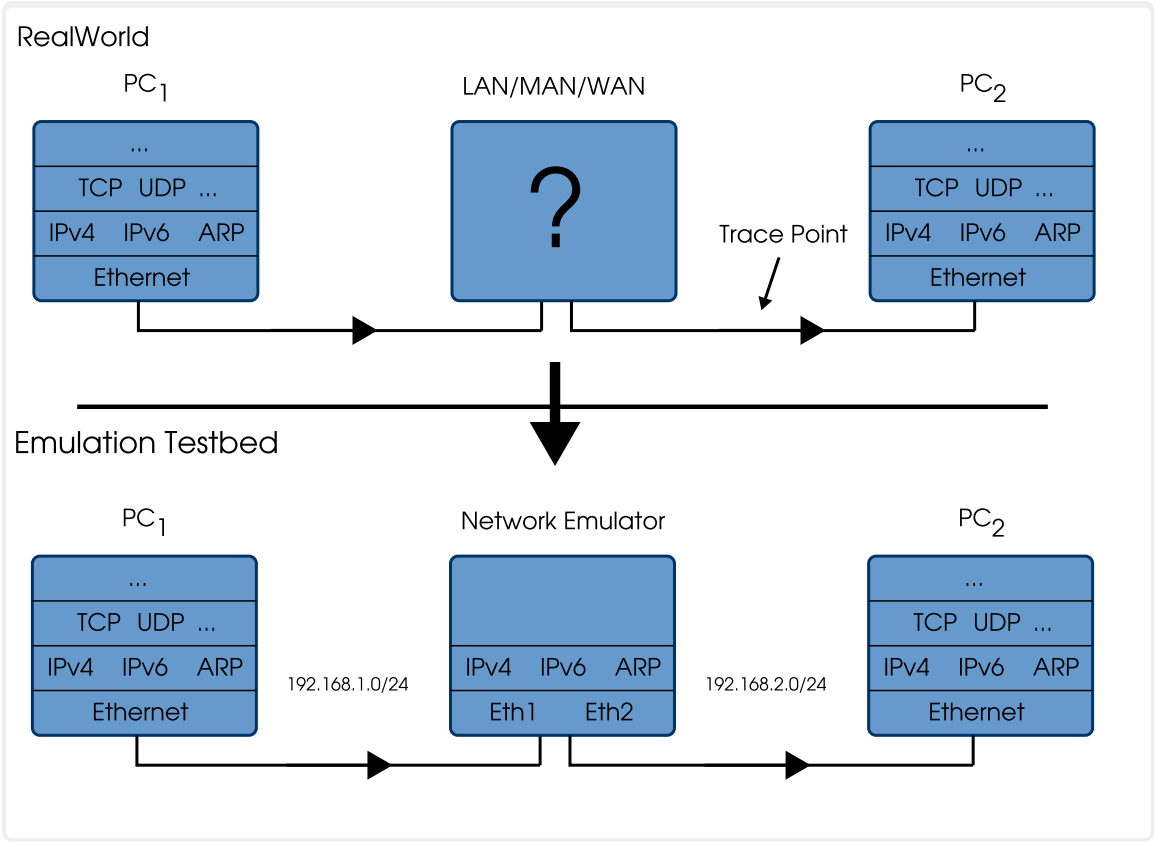
- `match [ u32 | u16 | u8 ] PATTERN MASK [ at OFFSET | nexthdr+OFFSET]`

- `iptables -A PREROUTING -t mangle -i eth0 -j MARK --set-mark 6`



Trace Based Emulation

- Using real world network characteristics
  - Analyze PCAP files
  - Ping target host/network
  - ttcp, iperf, netperf, netsend, ipproof, ...



Trace Based Emulation

- Based on distribution table feature
- Collect data to characterize distribution (e.g. ping your target and cut RTT data)
- Example: pings taken on 02.01.2012 by train from Berlin/Muc to server
  - `ping -c 10000 jauu.net > ping-data.raw`
  - `cat ping-data.raw | grep icmp_seq | cut -d"=" -f4 | cut -d" " -f1 > ping.dat`
- iproute2/netem/stats ping.dat
  - `mu: 1570.039912` (average)
  - `sigma: 2462.728332` (variation)
  - `rho: 0.888526` (distribution of RTT over time)
- Generate distribution table:
  - `iproute2/netem/maketable ping.dat > jauu.dist`
  - `cp jauu.dist /usr/lib/tc`

Packet Classification

- `tc qdisc del dev eth5 root 2>/dev/null`
- `tc qdisc add dev eth5 root handle 1: prio bands 2 pri-omap 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0`
- `tc qdisc add dev eth5 parent 1:1 handle 10: sfb`
- `tc qdisc add dev eth5 parent 1:2 handle 20: netem delay 1ms`
- `iptables -t mangle -A POSTROUTING -o eth5 -p tcp --dport 5001 -j CLASSID --set-class 1:2`

Links

- http://www.formann.de/wp-content/uploads/2011/04/Layer_2_Link_Emulation_in_virtuellen_Netzen.pdf
- http://staff.science.uva.nl/~delaat/rp/2010-2011/p32/report.pdf

Thank You!

Contact

- Hagen Paul Pfeifer
- hagen.pfeifer@protocollabs.com
- Key-Id: 0x98350C22