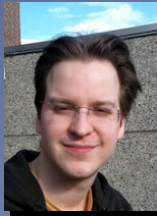


SliceTime

A platform for accurate and scalable network emulation



Elias Weingärtner



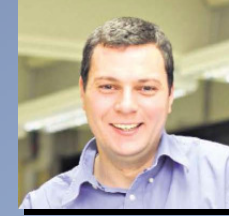
Florian Schmidt



Hendrik vom Lehn



Tobias Heer

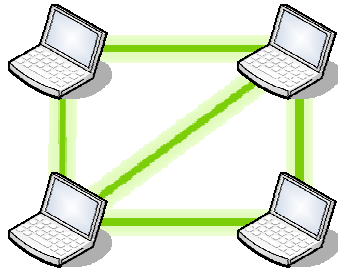


Klaus Wehrle

<http://comsys.rwth-aachen.de/>

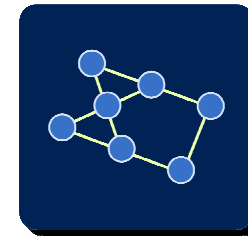
Aachen, 14.7.2011

How to evaluate networking software at large scale?



Network Testbeds

Drawbacks: Scalability & Cost



Network Simulation

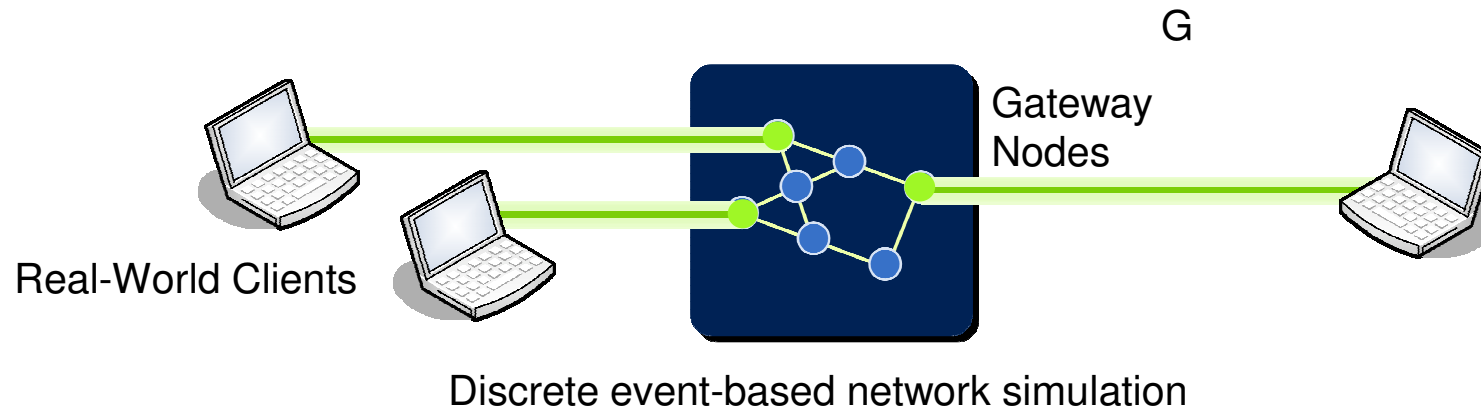
Models instead of software, no operating system...



Network Emulation

Requires real hardware and real simulations

**SliceTime solves
this problem**



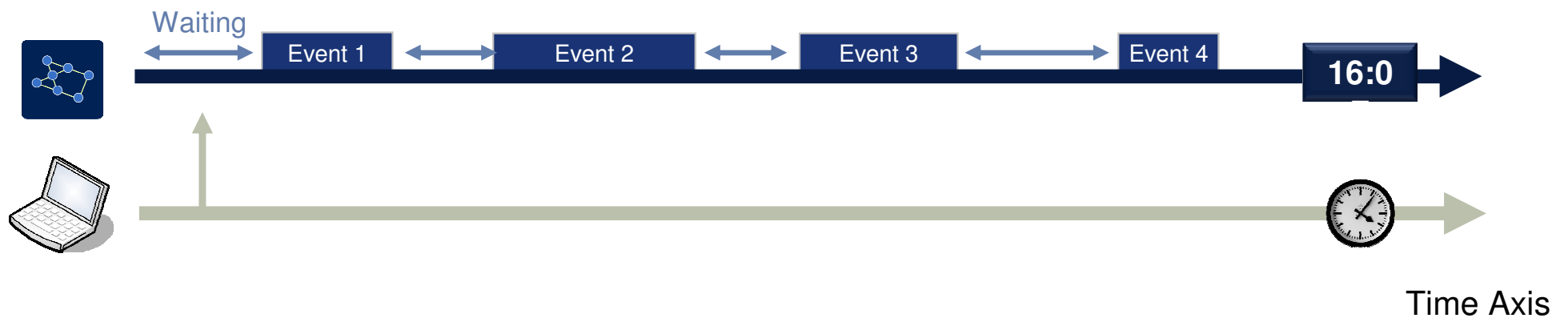
- **Real-World clients**

- ▶ Execute communications software & operating system

- **Discrete event-based network simulator**

- ▶ Models interconnecting network
- ▶ Examples: ns-2, ns-3, OMNeT++
- ▶ Also provides simulated hosts → scalability
- ▶ Simulated environment: virtual mobility, radio propagation...

- **Different timing concepts**
 - ▶ Network simulation: series of discrete events
 - ▶ Real-world clients: continuous wall-clock time
- **Current common solution**
 - ▶ Pin simulation events to wall-clock time
 - ▶ Wait between events



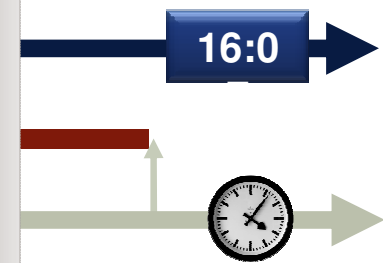
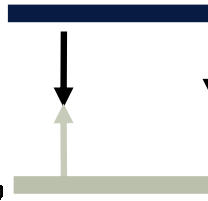
Time Drifting Issue

- **Problem: Many Simulations are not real-time capable**
 - ▶ Computationally complex models
 - ▶ Many simulated nodes
- **Simulation is overloaded → time drift**

- **Incorrect Results**

- ▶ Expiration of

```
elias@elias-VirtualBox: ~  
File Edit View Search Terminal Help  
PING 10.1.1.3 (10.1.1.3) 56(84) bytes of data.  
64 bytes from 10.1.1.3: icmp_req=1 ttl=64 time=123 ms  
64 bytes from 10.1.1.3: icmp_req=2 ttl=64 time=197 ms  
64 bytes from 10.1.1.3: icmp_req=3 ttl=64 time=281 ms  
64 bytes from 10.1.1.3: icmp_req=4 ttl=64 time=360 ms  
64 bytes from 10.1.1.3: icmp_req=5 ttl=64 time=423 ms  
64 bytes from 10.1.1.3: icmp_req=6 ttl=64 time=498 ms  
64 bytes from 10.1.1.3: icmp_req=7 ttl=64 time=564 ms  
64 bytes from 10.1.1.3: icmp_req=8 ttl=64 time=639 ms  
64 bytes from 10.1.1.3: icmp_req=9 ttl=64 time=713 ms  
64 bytes from 10.1.1.3: icmp_req=10 ttl=64 time=790 ms  
64 bytes from 10.1.1.3: icmp_req=11 ttl=64 time=867 ms  
64 bytes from 10.1.1.3: icmp_req=12 ttl=64 time=929 ms  
64 bytes from 10.1.1.3: icmp_req=13 ttl=64 time=1004 ms  
64 bytes from 10.1.1.3: icmp_req=14 ttl=64 time=1079 ms  
64 bytes from 10.1.1.3: icmp_req=15 ttl=64 time=1165 ms  
64 bytes from 10.1.1.3: icmp_req=16 ttl=64 time=1233 ms  
64 bytes from 10.1.1.3: icmp_req=17 ttl=64 time=1303 ms  
64 bytes from 10.1.1.3: icmp_req=18 ttl=64 time=1411 ms  
^C  
--- 10.1.1.3 ping statistics ---  
32 packets transmitted, 18 received, 43% packet loss, time 3242ms  
rtt min/avg/max/mdev = 123.913/754.997/1411.289/384.271 ms, pipe 14  
elias@elias-VirtualBox:~$ sudo ping -i 0.1 10.1.1.3
```

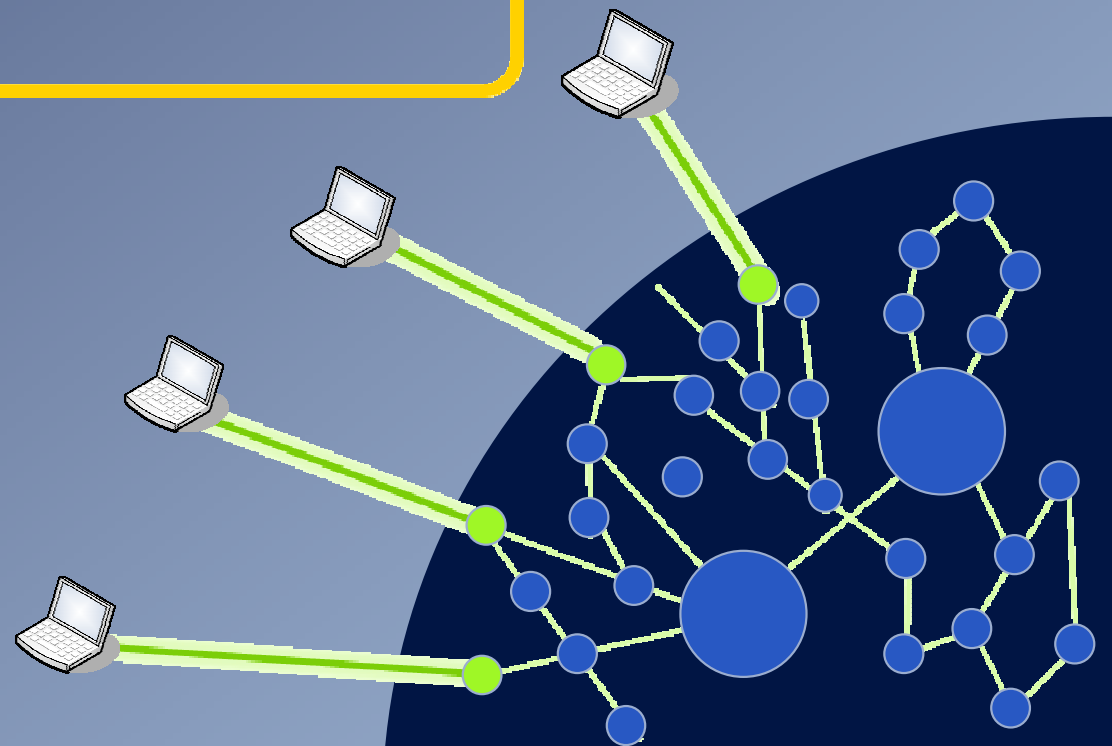


How can time drifting be prevented to enable large-scale and complex network emulation scenarios?

Two options:

1. Make the simulation fast enough

2. Slow down the real clients to match the simulation's speed



1. Tight synchronization of clients and simulation

- ▶ Limit drifting to 1ms or less (for WAN scenarios)

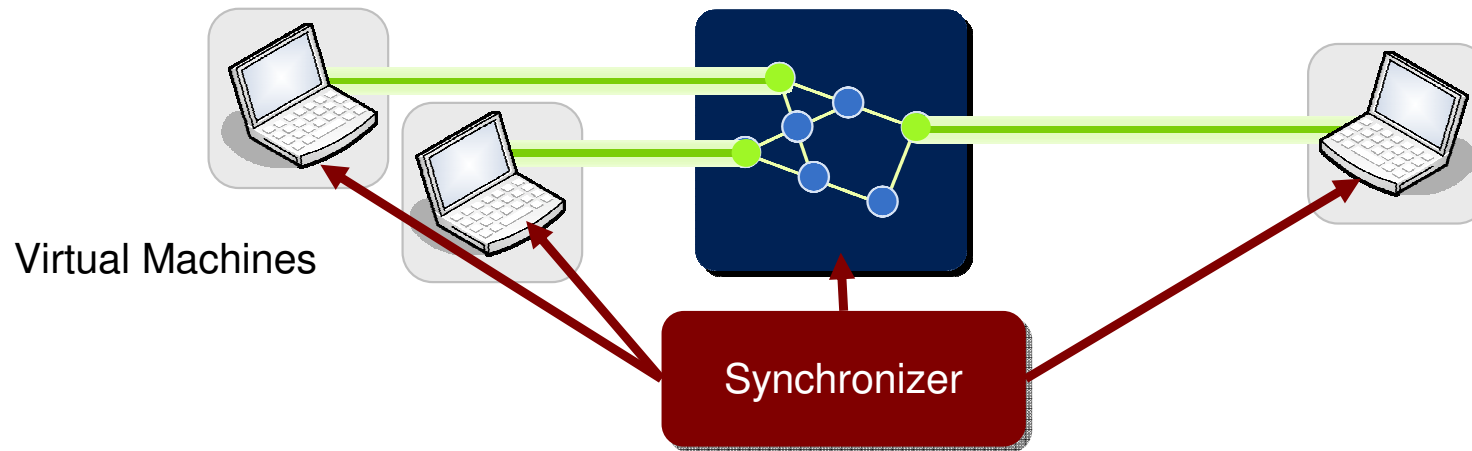
2. Need to slow down real-world software clients

- ▶ Unmodified communications software
- ▶ Legacy operating systems (Linux or Windows)
- ▶ Slow down must be transparent to the clients
→ provision of virtual time

3. Little overhead due to synchronization

- ▶ Additional run-time
- ▶ Additional delays or measurement artifacts

SliceTime: A Synchronized Network Emulation platform



- **Synchronizer**

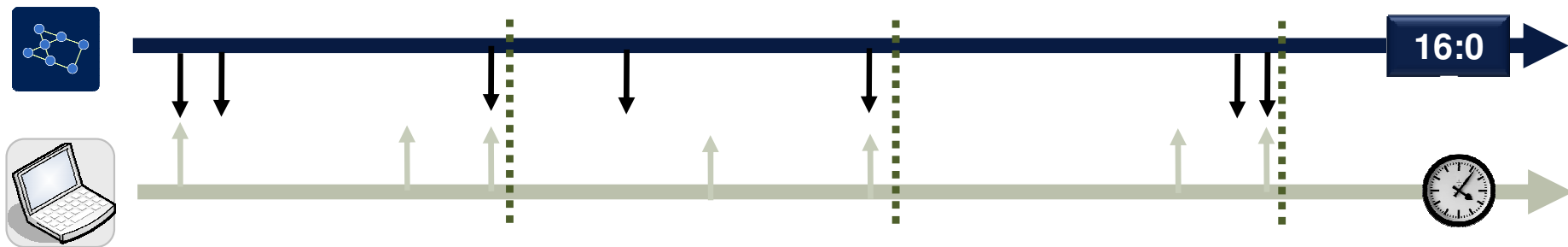
- ▶ Synchronization algorithm aligns execution of clients and simulation

- **Virtual machines provide needed level of control**

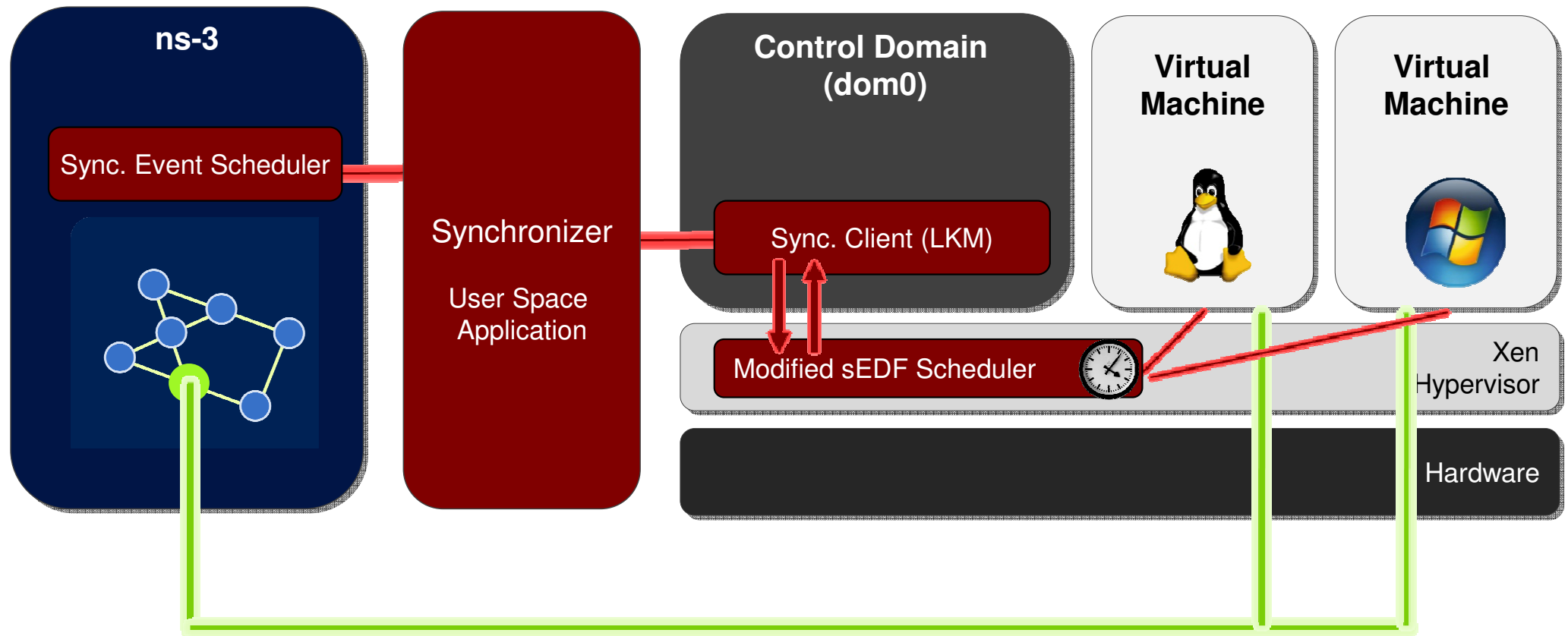
- ▶ Control over run-time behavior
- ▶ Full control over system context/timers → provision of virtual continuous time

Synchronization Algorithm

- **Goal: Limit time drifting**
 - ▶ No assumptions about future run-time behavior
 - ▶ No snapshotting & rollbacks
- **Barrier Algorithm**
 - ▶ Assign slices of run-time
 - ▶ Blocking at end of time slice
 - ▶ Clients notify synchronizer after they have finished
- **Synchronization accuracy corresponds to time slice size**



SliceTime Implementation



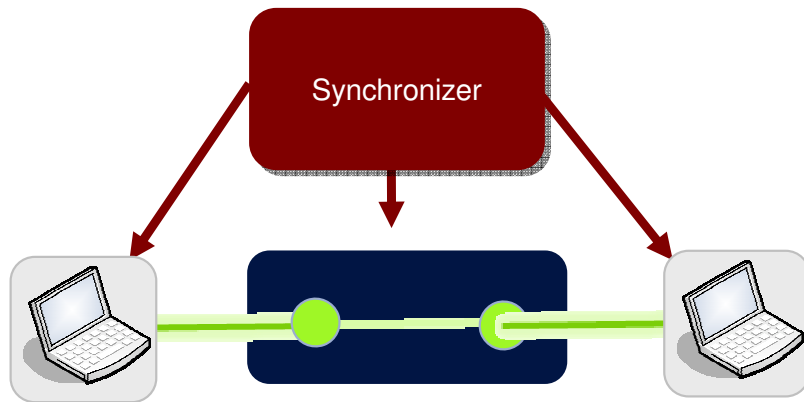
Data Communication Flow
•Tunneled Ethernet Frames
•802.11 Frame Tunnel

Evaluation

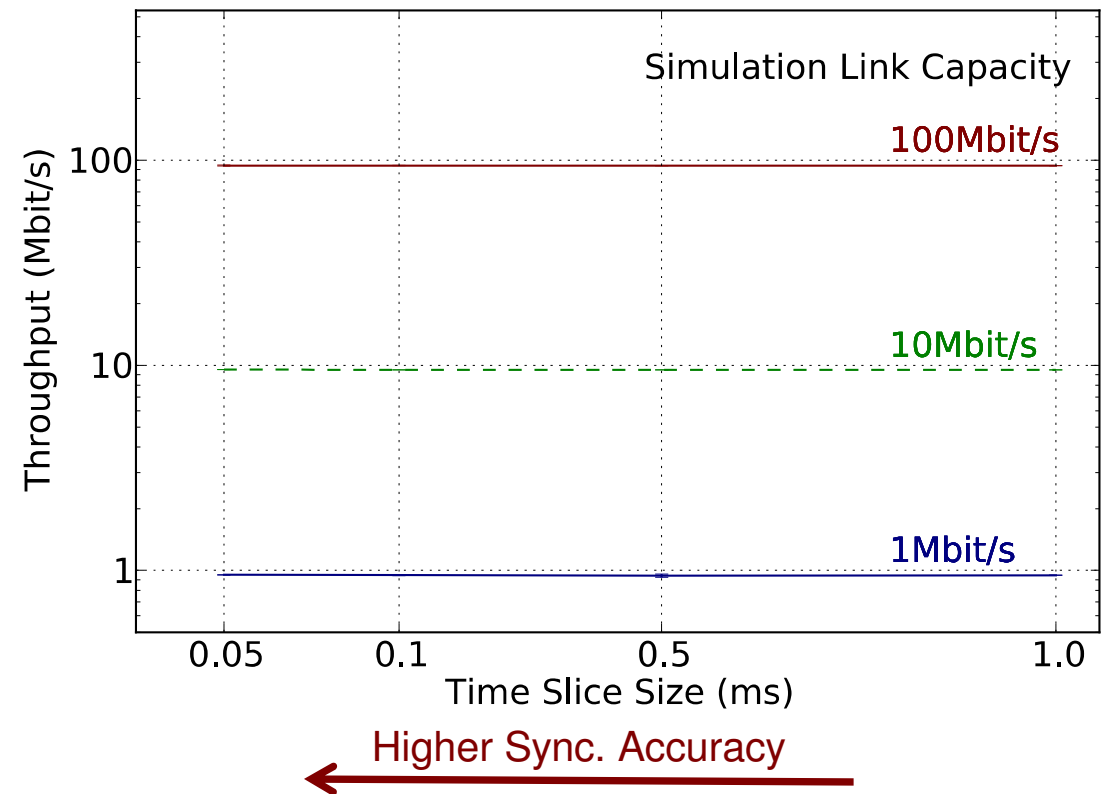
How accurate is SliceTime?

How much overhead is caused by the synchronization?

How is network throughput affected by time slice size?

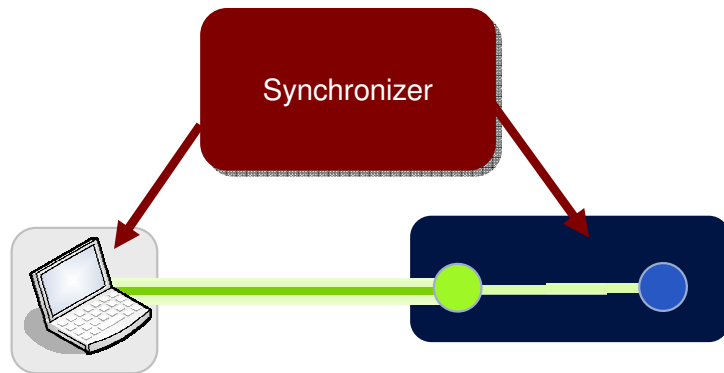


Measurement: netperf TCP_STREAM benchmark
Different levels of sync. accuracy

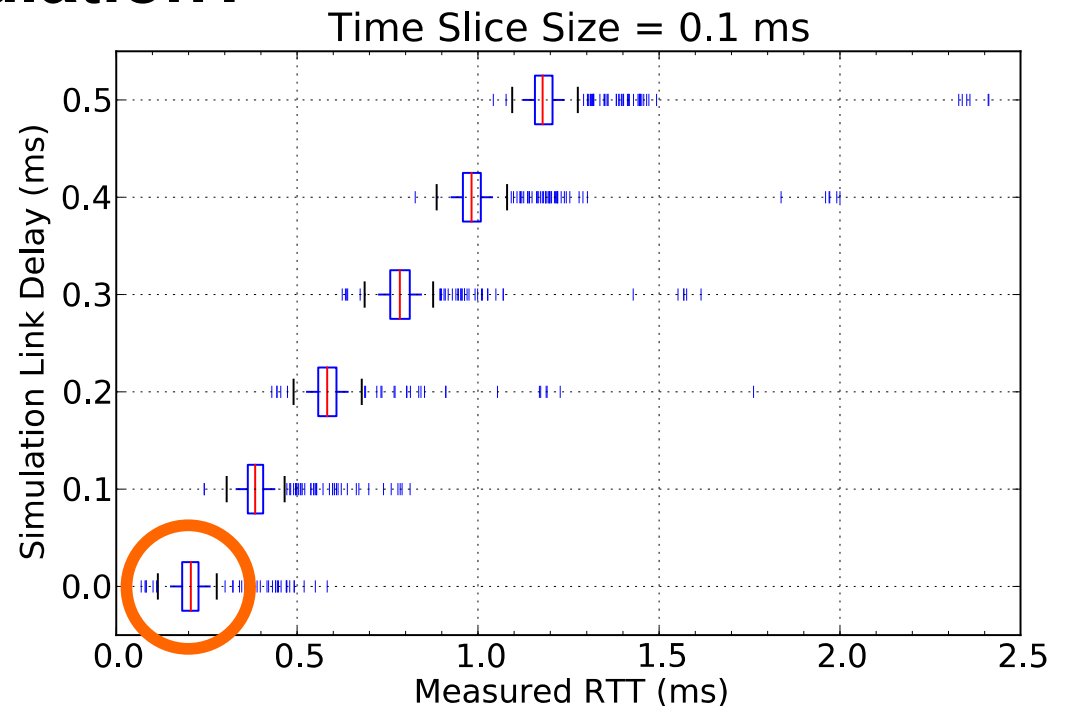


- **Perceived bandwidth is invariant to time slice size**

How accurate is the time integration of VMs and the simulation?

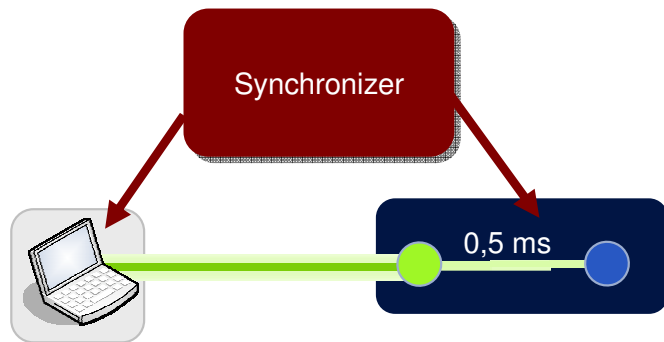


Measurement: 1500 RTTs (ICMP Echo Replies)
Simulated Link Delays between: 0,0 – 5ms
Static time slice size of 0.1 ms



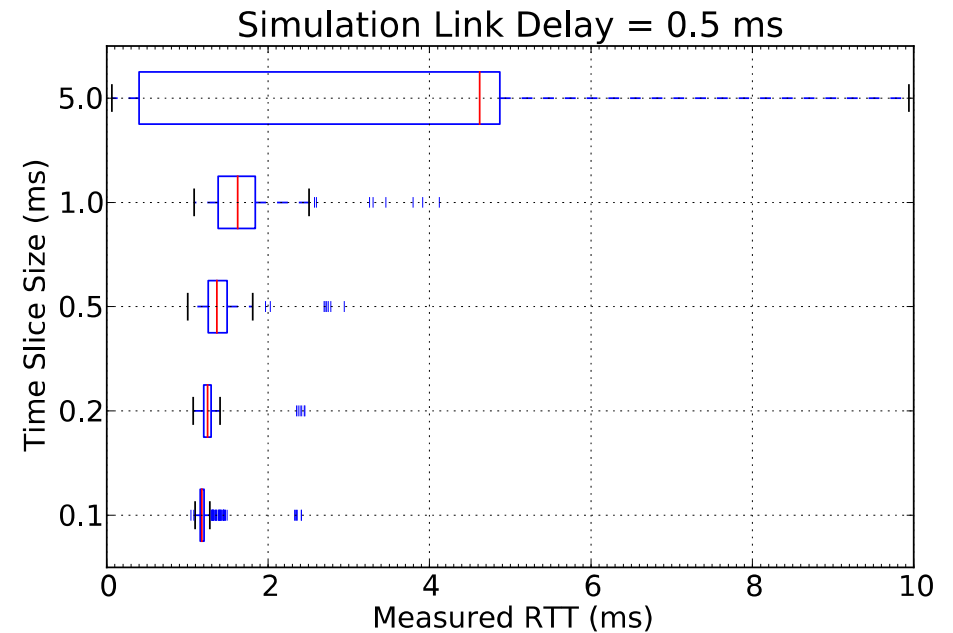
- **If no simulation delay is present → RTTs around ~ 0.2ms**
 - ▶ Base delay: Time needed for data exchange between VM & sync
- **RTT distributions shifted by twice simulation delay**

How do different time slice sizes influence the results?



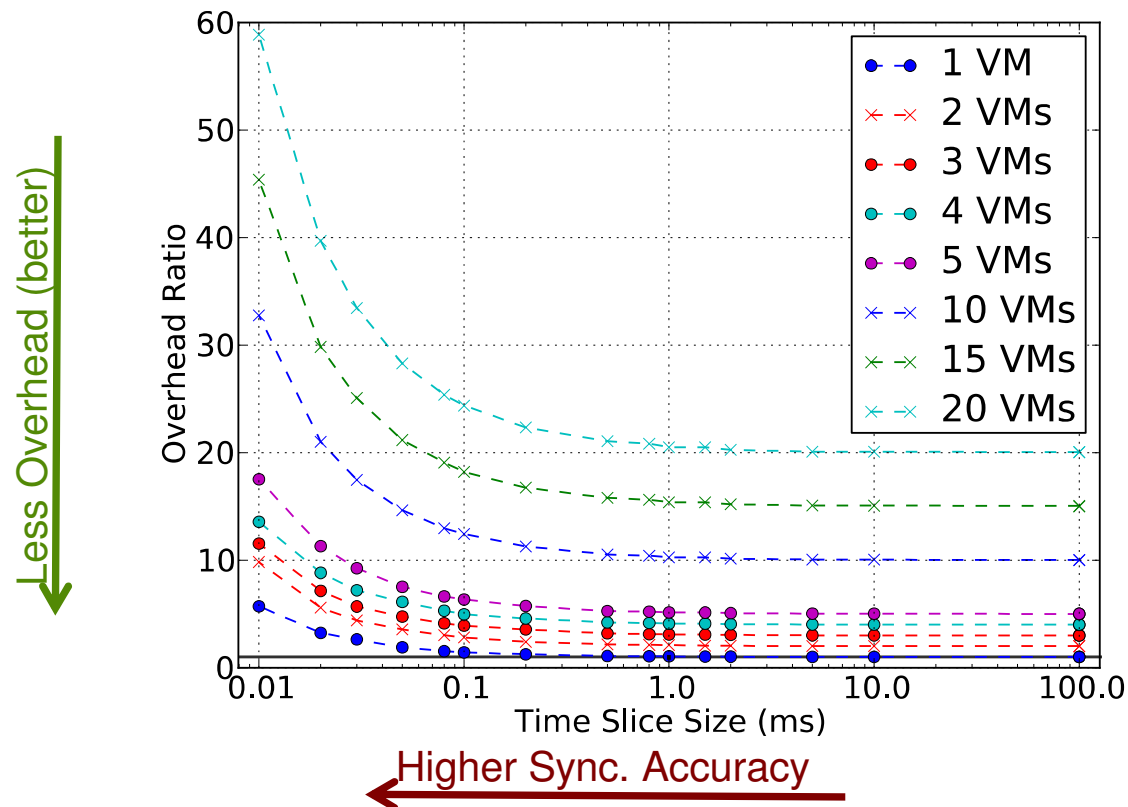
Measurement: 1500 RTTs (ICMP Echo Replies)
Variation: Time Slice Sizes

Higher Sync. Accuracy



- **RTT distributions converge to base delay for smaller time slices (higher accuracies)**

How long does it take to execute 1s of virtual time?



- **Synchronization introduces additional run-time overhead**
 - ▶ Less than 5% for time slices $> 0,5\text{ms}$
 - ▶ Linear in the number of VMs

- **SliceTime allows network emulation scenarios with network simulations of any run time behavior**
- **SliceTime is accurate regarding timing and throughput**
- **SliceTime is resource efficient**
 - ▶ Low overhead even for time slices less 1ms
 - ▶ Saves physical hardware resources in comparison to real test beds
- **SliceTime is open source**
 - ▶ Get it at <http://www.comsys.rwth-aachen.de/projects/slicetime>
- **SliceTime extends the applicability of network emulation**

Questions?

Implementation

Two main tasks:

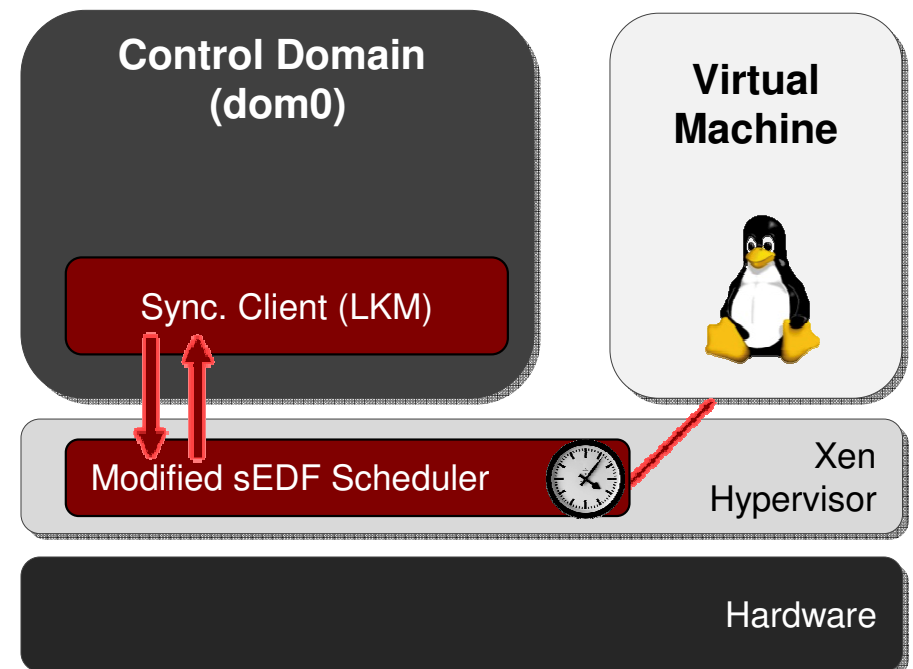
- Execute VMs & simulation for exact time slice duration
- Provide VMs with illusion of gapless virtual time

- **Synchronization Client**

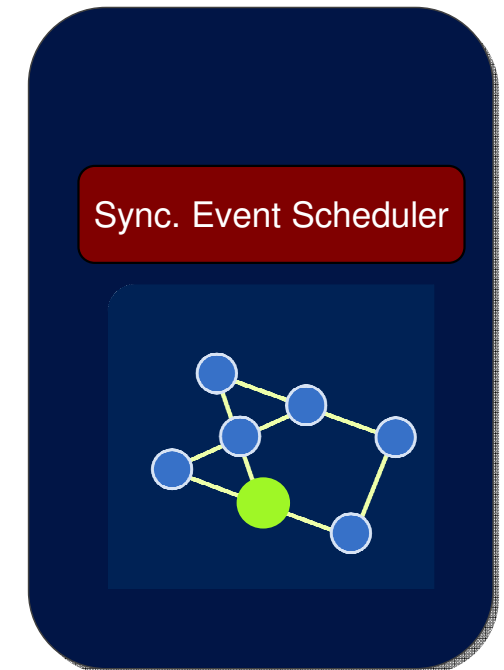
- ▶ Linux Kernel Module → save context switches

- **Modified sEDF scheduler**

- ▶ Execute Xen domains for time slice duration
 - Extra scheduling queue for synchronized domains
 - Self-correction mechanism to overcome misattribution of run-time
- ▶ Virtualizes time progression for synchronized domains
 - Calculates delta values for timers and clock sources



- **Synchronized Event scheduler**
 - ▶ Synchronizes any ns-3 simulation with synchronizer/VMs
 - ▶ Checks if next event in queue resides in current time slice
- **Different ns-3 extensions**
 - ▶ Tunnel protocol → data exchange with VMs
 - ▶ WiFi emulation extensions
 - Provides VMs with wireless networking interface
 - Interface is intergrated with 802.11 model of ns-3



- **Implements barrier synchronization algorithm**
 - ▶ Assignment of time slices
 - ▶ Synchronizes multiple VMs with multiple simulations
- **User-space application**
 - ▶ Can run on VM, simulation slave or dedicated host
 - ▶ Lightweight signaling protocol
- **VMs and simulations may join sync. dynamically**
 - ▶ Allows VM bootstrapping out of synchronization

Synchronizer

Can SliceTime ease the evaluation of networking software?

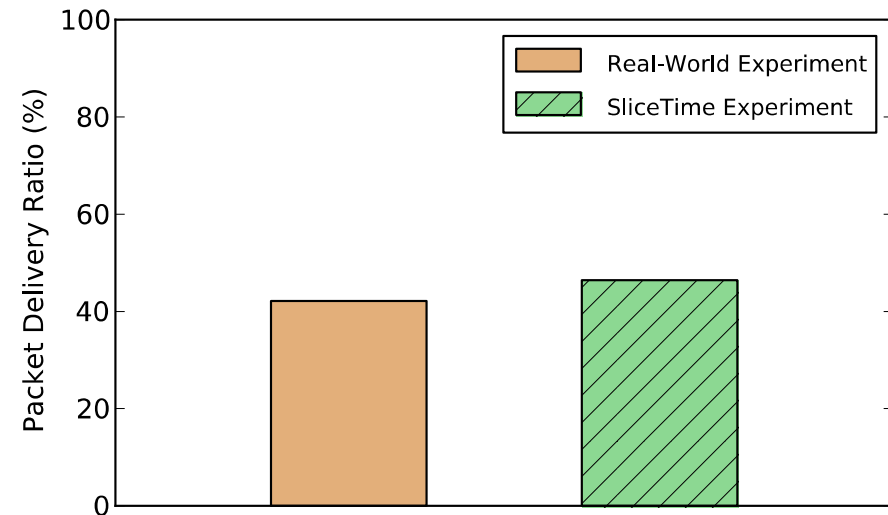
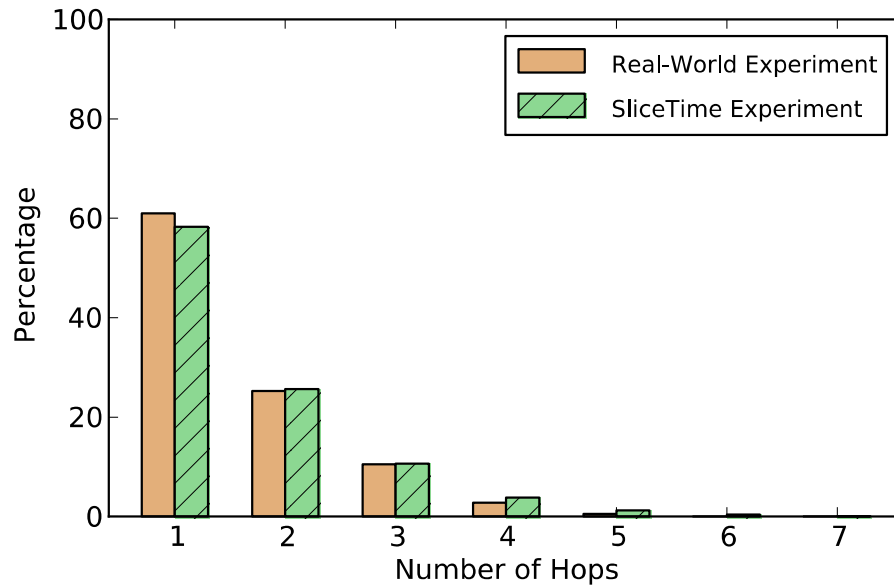
AODV Experiment (Gray et al, 2003)

- 33 laptops running AODV
- 40 people carrying them around (on an athletic field)
- Random UDP traffic
- Laptops log traffic + position (GPS)
 - Logs available at CRAWDAD

The SliceTime equivalent

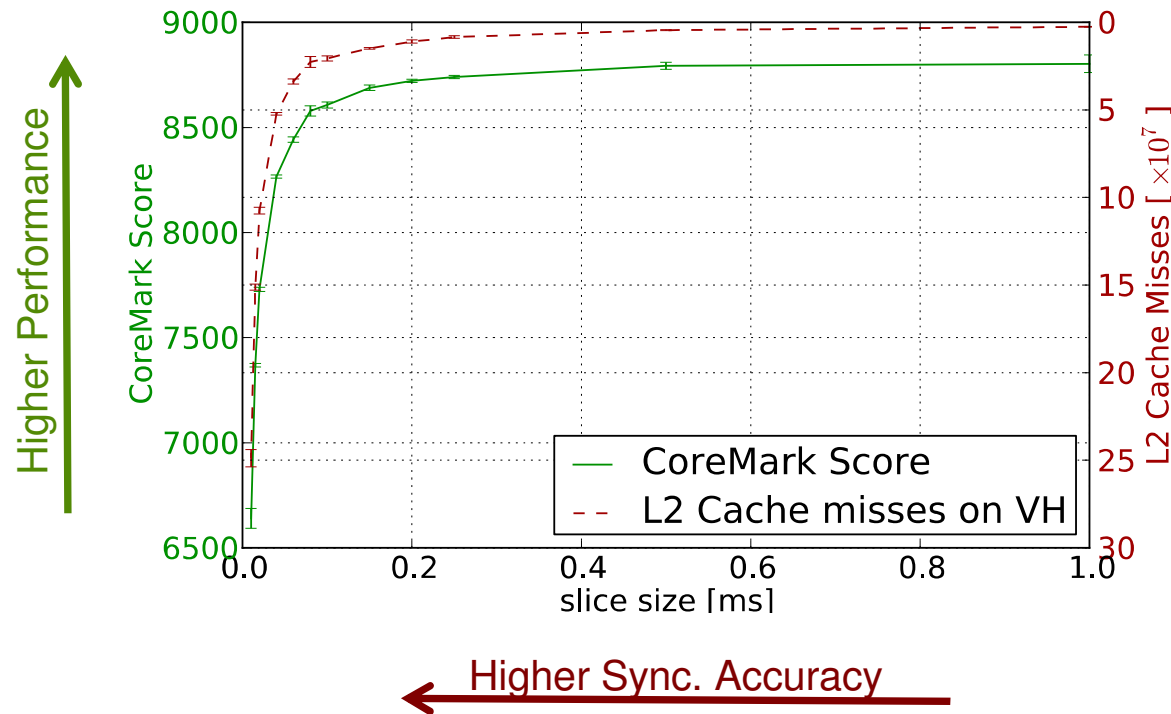
- 33 Xen HVM domains / AODV
- SliceTime 802.11 extensions
- 1 physical PC
- Ns-3 mobility model based on GPS traces
- Traffic generator

How do the results compare?



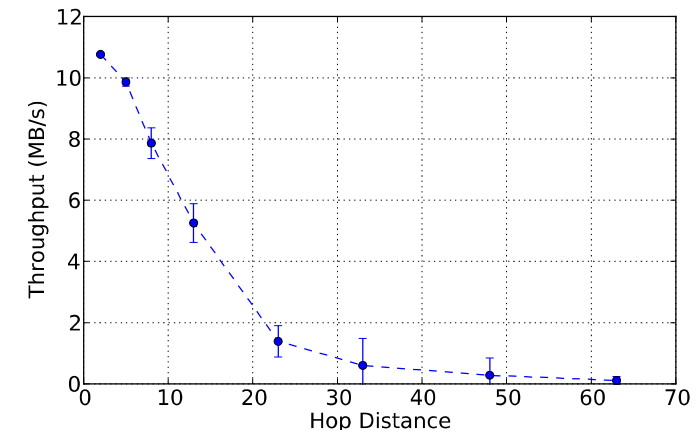
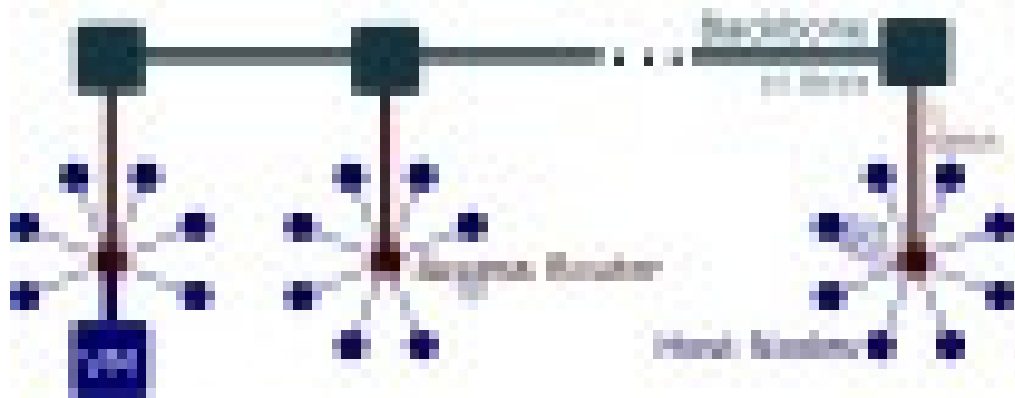
- **SliceTime produces results close to real-world measurements**
- **Always differences due to real-world/simulation disparity**

How about the CPU performance? Doesn't the synchronization cause artifacts?



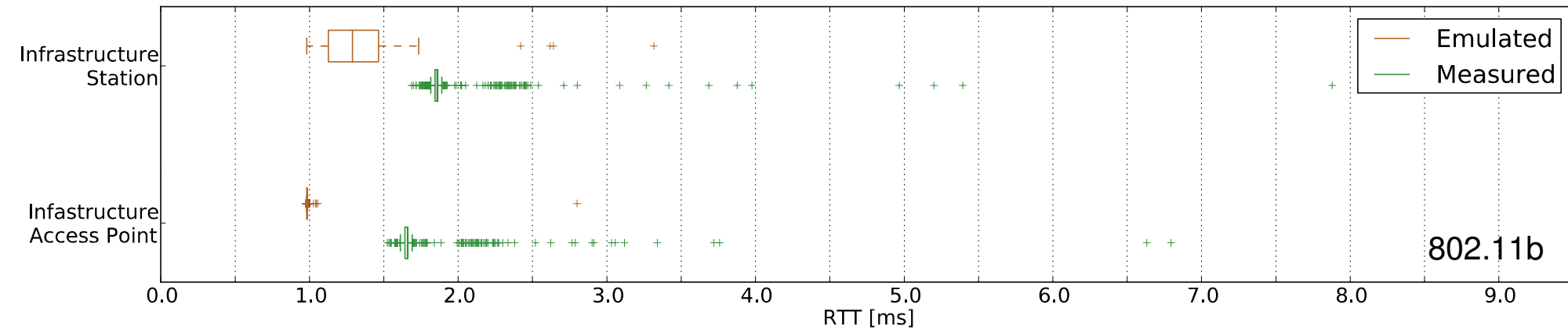
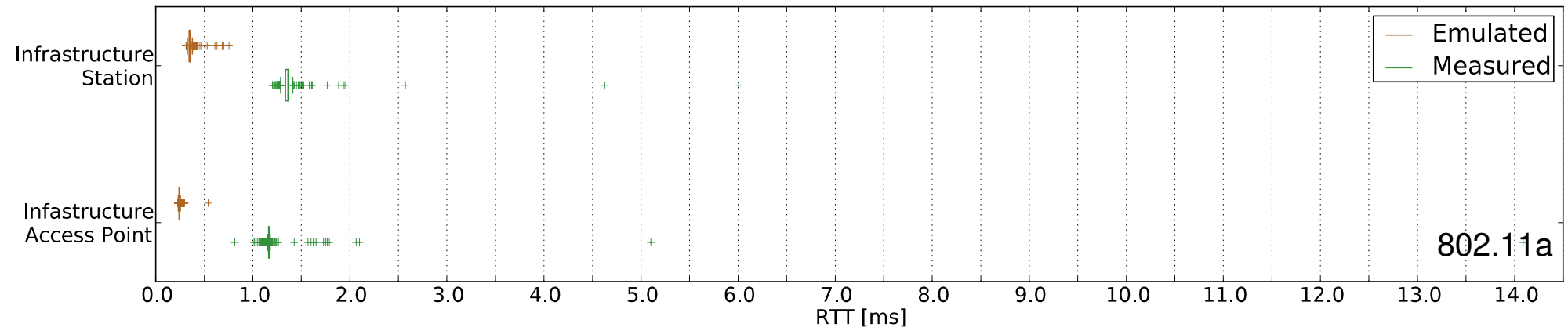
- **CoreMark score decreases for small time slices**
 - ▶ Almost no impact for slices greater than 0.1 ms
 - ▶ Explanation: More L2 cache misses

SliceTime Simulation scalability



- **Setup: 15000 simulated nodes (60 stars with 250 nodes)**
 - ▶ Exchange data blocks among each other using HTTP
 - ▶ Executes ~15 times slower than real-time
 - ▶ 1 VM attached to backbone
- **HTTP performance measured with curl**
 - ▶ Expected result

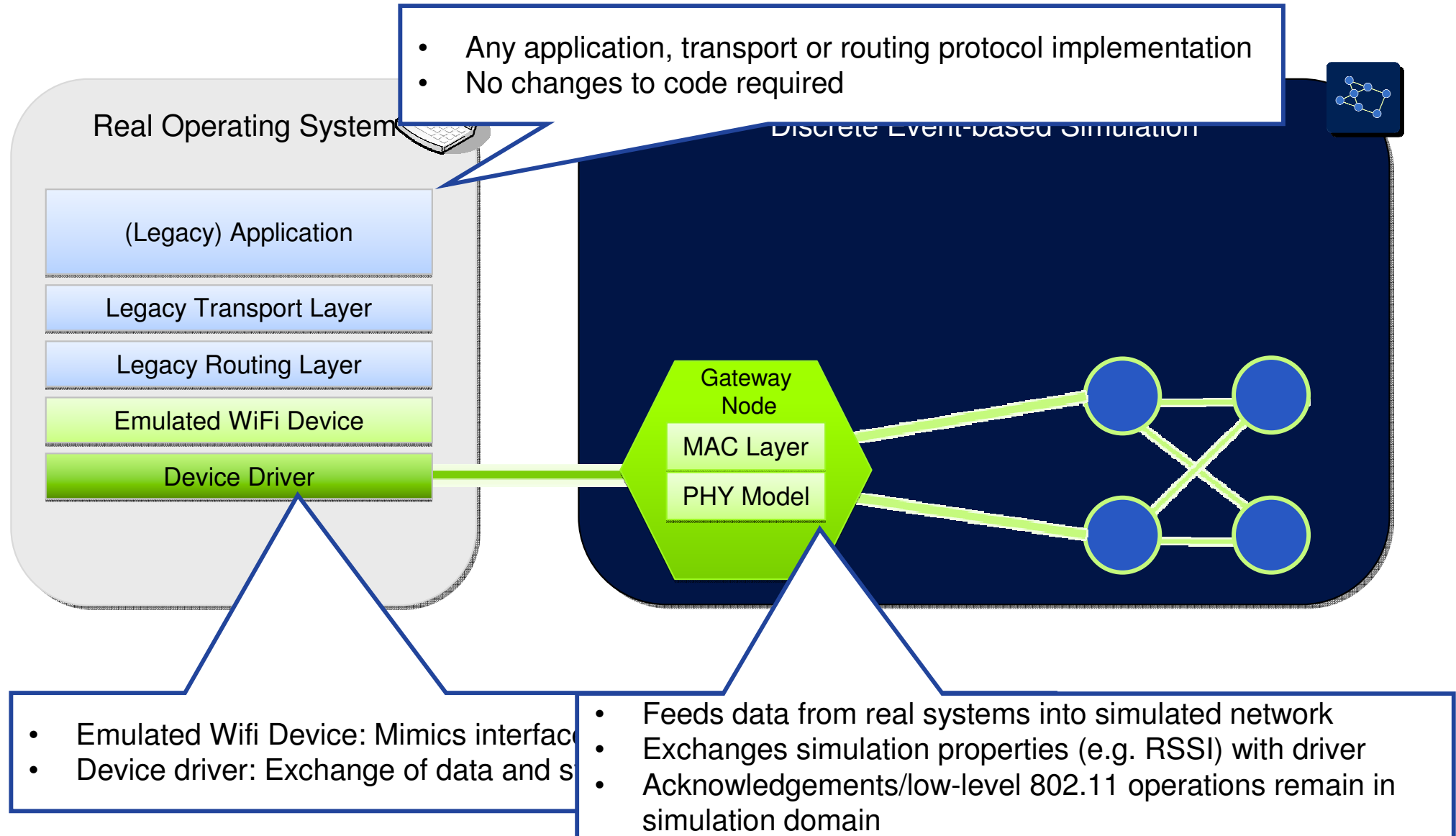
How do round trip times compare to real world 802.11?



- **Emulated RTTs are lower than real world measurements**

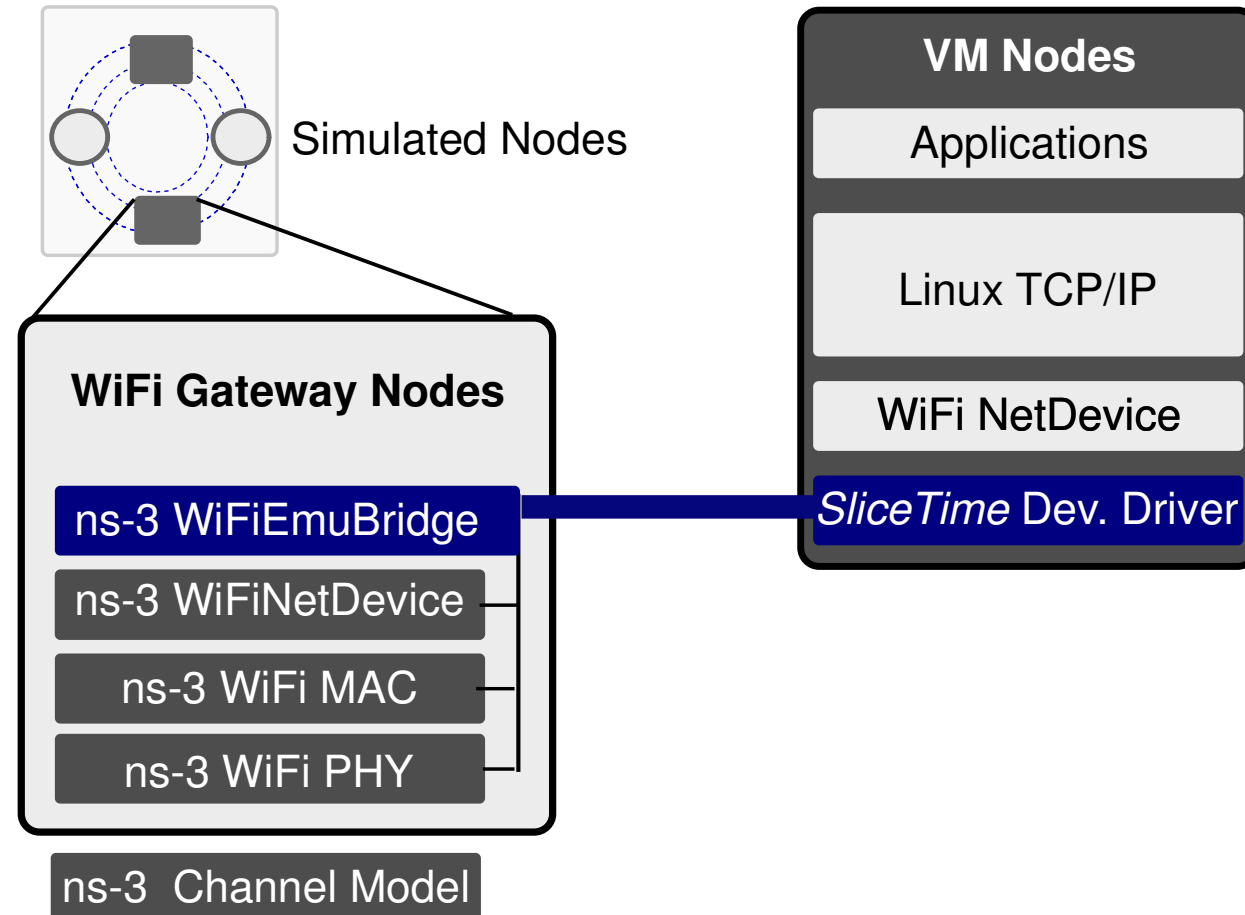
► ns-3 only approximations for link-level delays; no system delays

Device Driver-enabled Wireless Network Emulation



SliceTime WiFi extensions

WiFi-Simulation



The screenshot shows the Wireshark network protocol analyzer interface. The main pane displays a list of captured packets, with packet 12 selected. The details pane below shows the structure of the selected packet, including the IEEE 802.11 Beacon frame and its management frame structure. The packet list table is as follows:

No.	Time	Source	Destination	Protocol	Info
9	19.999977	00:00:00:00:00:02	ff:ff:ff:ff:ff:ff	IEEE 802	Beacon frame, SN=39, FN=0, Flags=0....., BI=35081,
10	22.499972	00:00:00:00:00:02	ff:ff:ff:ff:ff:ff	IEEE 802	Beacon frame, SN=40, FN=0, Flags=0....., BI=35081,
11	24.999979	00:00:00:00:00:02	ff:ff:ff:ff:ff:ff	IEEE 802	Beacon frame, SN=41, FN=0, Flags=0....., BI=35081,
12	27.499978	00:00:00:00:00:02	ff:ff:ff:ff:ff:ff	IEEE 802	Beacon frame, SN=42, FN=0, Flags=0....., BI=35081,
13	29.999980	00:00:00:00:00:02	ff:ff:ff:ff:ff:ff	IEEE 802	Beacon frame, SN=43, FN=0, Flags=0....., BI=35081,
14	32.499979	00:00:00:00:00:02	ff:ff:ff:ff:ff:ff	IEEE 802	Beacon frame, SN=44, FN=0, Flags=0....., BI=35081,
15	34.999978	00:00:00:00:00:02	ff:ff:ff:ff:ff:ff	IEEE 802	Beacon frame, SN=45, FN=0, Flags=0....., BI=35081,

The details pane for packet 12 shows the following structure:

- Data Rate: 1.0 Mb/s
- Channel frequency: 2412 [BG 1]
- Channel type: 802.11b (0x00a0)
- SSI Signal: -60 dBm
- SSI Noise: -101 dBm
- IEEE 802.11 Beacon frame, Flags: 0.....
- IEEE 802.11 wireless LAN management frame

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 00 00 18 00 6f 00 00 00 e0 2e 42 06 00 00 00 00  ....0... ..B....
0010 00 02 6c 09 a0 00 c4 9b 80 80 00 00 ff ff ff ff  ..l.....
0020 ff ff 00 00 00 00 02 00 00 00 00 02 a0 02  ....
0030 00 00 00 00 06 42 2c 40 09 89 00 00 00 0c 77 69  ....B,@ .....wi
```

The status bar at the bottom indicates: wemu0: <live capture in progress... Packets: 26 Displayed: 26 Marked: 0 Profile: Default

- **Wireshark for live monitoring of simulated WiFi networks**
 - ▶ Inspection of low-level 802.11 properties using Radiotap headers

```
Kismet Sort View Windows
Name          T C Ch Pkts Size                               Kismet_200
Network-1     A N 3  135 4K
Network-3     A N 3  214 4K
BSSID: 00:00:00:00:00:06 Last seen: Nov 6 19:25:27 Crypt: None Manuf: Unknown Elapsed
Network-4     A N 7  526 5K                                00:08,40
Network-5     A N 7  483 4K                                Networks
Network-6     A N 7  450 4K                                9
Network-7     A N 9  187 4K                                Packets
Network-8     A N 9  100 5K                                2754
MAC           Type   Freq Pkts Size Manuf
00:00:00:00:00:06 Unknown 2422 195 1K Unknown Pkt/Sec
00:00:00:00:00:07 Wireless 2422  19  2K Unknown 0
Filtered
0
No GPS info (GPS not connected) Pwr: AC
42
0
Data
INFO: Welcome to the Kismet Newcore Client... Press '' or '' to activate menus.
INFO: Connected to Kismet server 'Kismet_2009'
INFO: Got configure event for client
INFO: Saved data files
INFO: Found IP range 192.168.0.28/255.255.255.254 for network 00:00:00:00:00:12
menu0
Hop
```

- **Kismet being executed in simulated network**
 - ▶ Allows the execution of unmodified legacy applications that make use of Linux Wireless Extensions