# Partov: A Network Simulation and Emulation Tool

Behnam Momeni and Mehdi Kharrazi

***Abstract*—Network protocols design and evaluation requires either full implementation of the considered protocol and evaluation in a real network, or simulating it based on a model. There is a middle approach in which both simulation and emulation are used to evaluate a protocol. In this manuscript the Partov engine is presented, which provides both simulation and emulation capabilities simultaneously.**

**Partov benefits from a layered and platform-independent architecture. As a pure simulator, it provides an extensible plugin-based platform and can be configured to perform both real-time and non real-time discrete-event simulations. It also acts as an emulator, making interaction with real networks possible in real-time. Additionally a declarative XML-based language is used, acting as a glue between simulation and emulation modules and plugins. It supports dynamic network modeling and simulation based on continuous time Markov chains. Partov is compared with other well known tools like NS-3 and real processes like Hping3. It is shown that Partov puts less overhead and is much more scalable than NS-3.**

***Index Terms*—Network Simulator, Network Emulator, Discrete Event Simulation, Plugin-based Extensible Infrastructure, Real-time Simulation**

## I. Introduction

The world of networking has greatly evolved over the past few decades and the protocols used have become more and more complex. While conducting research on network protocols, an instant of time comes at which it is required to examine the consequences and impacts of several design decisions or evaluate the correctness and performance bounds of the proposed protocol. This can be accomplished by implementing all possible aspects of the protocol and evaluating the resulting implementation on an actual network topology using real hardware. But such approach would be time consuming and costly as it would require a full and complete implementation of the protocol and extensive hardware.

It is then that simulators become quite valuable tools as they provide a relatively easy and fast way towards evaluating the protocol under study. By employing a simulator the researcher can simplify the implementation process, avoiding full implementation and deploying arbitrary topologies without requiring physical devices. Nevertheless, this abstraction separates obtained results from absolute reality which is a shortcoming with all simulators (i.e. due to limitations of the simulation models). On the other hand, networks such as the Internet are large, dynamic, and complex systems which can not be fully modeled. Hence, even though the simulator could simulate a protocol, it can not fully take into account the effects of the protocol and network on each other.

B. Momeni is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
E-mail: b_momeni [AT] ce.sharif.edu

M. Kharrazi is with the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
E-mail: kharrazi [AT] sharif.edu

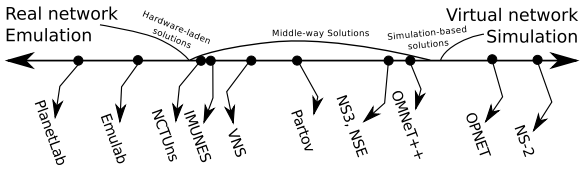



Fig. 1. Spectrum of network simulation/emulation tools

So there will be a trade-off between accuracy (and having more details) and early results (and simulating abstract models of reality). This trade-off forms a spectrum of solutions, as illustrated in Figure 1. In one extreme of the spectrum, hardware-laden solutions (*PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services* 2012; *Emulab - Network Emulation Testbed Home* 2014) sit which are of course more costly and troublesome to be established. In the other extreme simulation-based solutions (Kaparti, 2005; *The Network Simulator - NS-2* 2011) sit, where their effectiveness is heavily dependent on the employed abstract model and how well the important details are considered in them. Of course, some solutions (Wang and Huang, 2012; Zec and Mikuc, 2004; Casado and McKeown, 2005; Henderson, 2010; Nethi et al., 2007; Varga and Hornig, 2008) within this spectrum can provide benefits of both sides.

Ideally in a middle-way solution, the protocol/system under research is simulated based on available models and other interacting components are emulated (keeping as much details as possible). Indeed as the evaluated protocols/systems become more complicated, they are more likely to imply non-obvious impacts which are not drawn in simulations due to the simulation models' applied abstraction. Emulated environment alleviates this situation by restoring some details without complicating the simulation's abstract model. This paper introduces the *Partov*, a middle-way solution which provides simulation and emulation capabilities and allows real-time synchronization/interaction of virtual and real worlds.

There are interesting applications possible with Partov and more generally the middle-way solutions. For example in order to evaluate the call performance in the *Skype* network, as studied in (Lisha and Junzhou, 2006), one can abstract out the network topology using a simulator, instead of testing *Skype* in different network topologies/scenarios, but can not replicate the closed *Skype* protocol in the simulation world. However, it's possible to install the *Skype* clients on a number of virtual machines and connect them to the simulated network by emulation. This methodology can be extended to any closed source component interacting with the network.

Similar to the above, one can also consider complex com-

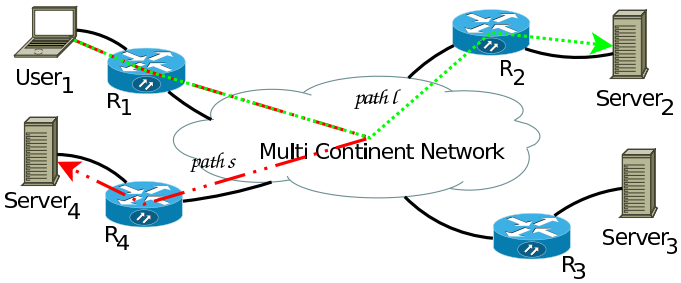



Fig. 2. Sample network of geographically distributed routers

ponents which interact with the simulated protocol/system, but are implemented independently. Consider designing a new geographical routing protocol, depicted in Figure 2, to route users towards the geographically nearest servers, as is common in content distribution networks (CDN). While the researcher concentrates more on the designed topology and protocol, he/she may miss side effects created by the design decisions. For example, the protocol may route all users through path $s$, due to shortest path selection rule, while in the real world other factors like high processing load in $R_4$ may make the total delay of longer paths, like the path $l$, less than the selected route.

More specifically, the total service time depends on many factors which generally can be divided to: 1) delay of the network's selected path which depends on the CDN's design, and 2) web server processing time, which depends on implementation of the web server (e.g. apache), given the workload conditions. Therefore, the independent part (i.e. the web server) will be deployed physically, avoiding abstraction of the web server, and interfaces with the simulated CDN.

In this paper an engine, namely *Partov*, from the middleway solutions will be discussed which enhances the interaction of virtual and real networks. The main contributions of this paper can be summarized as follows:

1) Providing a network simulation engine with increased extensibility via its plugins infrastructure,
2) Providing a portable emulation method allowing the simulated network to be placed on any machine and interface with the real network as needed,
3) Introducing a declarative network modeling language which allows one to describe a dynamic network (see Section III-E),
4) Achieving more reliable experimental results via coordination of simulation and emulation capabilities,
5) Providing an objective analysis of requirements that a simulation/emulation tool is required to provide.

In the rest of this paper, Section II proceeds with an objective analysis of requirements a simulation/emulation tool should satisfy. Afterward, Section III describes the Partov system, its architecture, and corresponding practical details. Section IV discusses test scenarios and evaluation of the proposed solution. Then Section V continues enumerating related works in the same direction, briefly discussing their features, and comparing Partov and other related solutions based on their capabilities. Finally, the manuscript is concluded in Section VI.

## II. Requirements Analysis

As with any tool development project, the work should be based on analyzing the required features for a network simulation/emulation tool. To that end, and in order to motivate the requirements, the CDN example which was noted earlier is reused. It's notable that this analysis and its results are not limited to the studied CDN example and could be motivated by any other generalized scenario.

In the noted CDN scenario, the goal is to design the network topology, best placement of the servers, and optimal mapping of the clients/servers pairs. The CDN can be separated in to the network, client, and server elements. Indeed the servers are real world implementations (e.g. apache) and won't change based on the network's and client's design. This requires the ability to simulate (`requirement R0`) networks and clients and interconnect them with the already implemented servers. Interconnection of these worlds requires an emulation capability; `requirement R1`.

The network part in turn consists of a number of routers distributed around the world. The router is a basic component and plays an important role in the network. So it's desired to have router simulation models ready out of the box. Having more pre-made simulation models (`requirement R2`) leads to less ad-hoc coding time. But each experiment may uncover some untouched areas which are unique to that experiment. For example, in the CDN design scenario, clients may be required to compose special web requests, their sizes obeying the Pareto distribution (Arlitt and Williamson, 1996) (considering the hosted pages) and send them at exponentially distributed (Arlitt and Williamson, 1996) intervals. Thus the simulation tool, while reasonably unable to support all models out of the box, must facilitate addition of new simulation models; `requirement R3`.

Afterward, both existing simulation models (e.g. routers) and newly implemented models (e.g. web clients) should be connected and form a network topology. The network topology is about what to do, in other words which network components with which interconnections, applications, etc. must exist, in contrast to how to do it. There are two general types of languages: 1) declarative languages, aimed at describing what should be done, and 2) imperative languages, aimed at dictating how each job should be performed. Considering the nature of network topologies, it's preferred to employ a declarative language for describing them; `requirement R4`.

The requirements elaborated so far are sufficient to make the simulation/emulation of a desired static topology possible. However, a lot of scenarios remain uncovered. In most real cases, some parts of network and/or its nodes may change dynamically. So being able to take these dynamic aspects into account adds a lot to the modeling flexibility. One of the useful concepts in modeling dynamic behaviors is Continuous Time Markov Chain (CTMC) (Ross, 2009, chap. 6). CTMC is capable of modeling many networking concepts. An example is the M/M/1 queue, a router receiving packets according to the Poisson distribution and serving them according to the Exponential distribution, which can be used in modeling of

navigation bursts in web traffic requests (Park et al., 2006). A complete simulation solution can not leave dynamic aspects and should provide some mechanism for simulating dynamic parts too; `requirement R5`.

Now consider the scenario in which interaction with some real applications may be required at different locations. For example, consider some thermal sensors distributed in an area, collecting temperature data at regular intervals. Each sensor can report its data to a stable super node close to it. Here it's desired, for example, to design the thermal sensors communication protocol. This can be accomplished by bringing generated events in a shared simulated network. So different communication scenarios can be tested without changing physical sensors themselves. For supporting such scenarios, the simulation tool requires a method for distributing the simulation at logically different locations and allowing distributed simulation models to gather events of interest from distributed real applications; `requirement R6`.

Next considerable fact, is about the deployment infrastructure. A suitable tool should not impose any restriction in this area. It's also preferred to support and facilitate different deployment scenarios. Starting from a simple personal computer till a powerful server shared within a laboratory and so on. Employing more widespread hardware infrastructures, motivates the need of multiuser support (`requirement R7`). Hence multiple users can connect to one installation and perform their own simulations and/or possibly share some simulation models.

The above requirements cover what a simulation/emulation tool should provide. And clearly all items should be satisfied at acceptable performance levels (`requirement R8`). For example while emulating a network, the virtual and real clocks must be kept synchronized. Or when a dynamic network is being simulated, changing factors should not have any effect on accuracy of packet loss, delay, or jitter performance metrics.

Having reviewed the requirements through different examples, in the next section Partov is introduced while noting how it satisfies the above stated requirements.

## III. The Partov Engine

*Portable And Reliable Tool fOr Virtualization (Partov)* system is built to answer a variety of needs as were elaborated in Section II. In answer to those requirements, the Partov system uses the architecture shown in Figure 3. The *Network Simulation Server (NSS)* in the central server consists of three subcomponents:

1) *Virtualization Engine* (see Section III-A) providing basic facilities (e.g. parsing packet headers) for implementing simulation/emulation models in response to `R0`, `R1`, and `R2` requirements,
2) *Plugin Infrastructure* (see Section III-B) which has been provisioned in response to needs and demand of extensibility (`R3`), and
3) *Simulation Server* which has the responsibility of connecting the centralized virtualization framework to the distributed frameworks (`R6`), identified as the *Client Framework (CF)* component (see Section III-C). It is also critical in managing multiple concurrent users (`R7`) through the *user management interface (UMI)*.

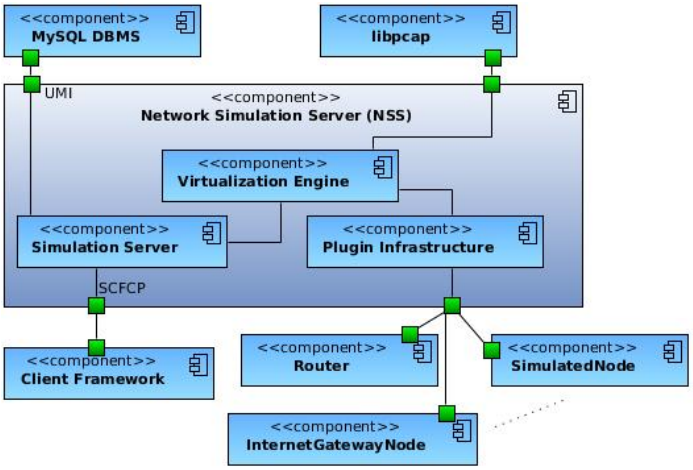



Fig. 3. The Partov System Component Diagram

These sub-components and their relationships are illustrated in Figure 3. Next a method is required for reusing and integrating simulation/emulation models. All models, pre-built and plugins, can be combined in arbitrary topologies (`R4`) via a declarative language (viz., *Partov Topology Language (PTL)*) which is discussed later in Section III-D. Lastly, for covering dynamic events (`R5`) and defining dynamic networks the *Continuous Time Markov Chain (CTMC)* concept is used (see Section III-E). Dynamic networks can be used in many cases like physical link failure, LAN topology changes (TC), node mobility, wireless varying background noise and so on. CTMC is described as a finite state machine in PTL, depicting Markovian states and transitions between them. So the *Virtualization Engine* component can control dynamic behavior by executing a CTMC model. Also the idea of a small and fast core *Virtualization Engine* is influential in bringing good performance (`R8`) for the system as a whole. In what follows each of the above noted components are discussed in detail.

### A. Virtualization Engine

The *Virtualization Engine* component is the heart of the central server which is responsible for virtualizing topology descriptions (`.map` files). It reads map files in the *PTL* format and creates *Object Oriented Networks (OON)* to be simulated, viz., one object is created for each virtual node, having a list of network interfaces objects, and one object is created for each virtual link.

To simulate a topology, an event-driven approach has been used. Link and interface objects *emit* signals to pass packets. All generated events are trackable by subscribing for emitted signals. For example, and in order to demonstrate the granularity of the generated events, a node which asks its interface object to send a packet, first emits a packet-sending signal. Upon receiving it, the link object emits a signal for propagating frame through other links and nodes. Third event is generated by destination's interface object which passes the packet to

node/plugin for processing. Thus, frames objects, which are indeed a chain of objects responsible for different network layers tasks, walk through the virtual topology while the minimum dependency is imposed between involved objects.

### B. Plugin Infrastructure

The core functionality of Partov can be extended by plugins. Each plugin is an encapsulated simulation and/or emulation module which is implemented based on the infrastructure provided by this component. Each plugin can use all of the utilities provided by the Partov for processing packets and can be configured using PTL parametric definitions. For example, routing functionality is implemented as the *Router* plugin, or the ability to connect to the real world and emulating packet sending is made available by the *InternetGatewayNode* plugin.

### C. Simulation Server

NSS can do its best for simulating a network on a single powerful server. But it may be desirable to distribute the simulation over several machines. Not only for better performance, but also it may be desired to integrate an independent application to participate in a simulation. This is where the *Client Framework (CF)* comes in handy. Using CF makes it possible to implement a simulation module, connect it to the central NSS component, and participate in the distributed simulation via a plug-and-play solution. Each simulated topology can select its own scheduling algorithm to be either real-time, in which simulation clock is kept in sync with wall clock, or compact-time, in which CPU is utilized completely and instead elapsed wall clock time can vary.

Each CF instance has a corresponding *remote agent* within the *Simulation Server* component. This agent is responsible for speaking the *Server-CF Communication Protocol (SCFCP)*, a TCP-based application layer protocol, in one distinct thread of execution. Also each topology in the server is scheduled independently in its separate thread of execution. Within each virtual topology, nodes and CF instances' remote agents work with the same priority and can register periodic and/or individual events with microseconds precision.

SCFCP performs three tasks: First, allows the user to instantiate/terminate virtual topologies remotely; emphasizing the role of Partov as a simulation server. Second, transfers all packets which are received by a virtual node to its corresponding CF instance and vice versa; so CF can act just like a plugin within the server. Third, allows a third party application to connect to different virtual topologies and to query their simulation states. As an example usage, one can implement a generic adapter by coding a CF which upon arrival of packets, writes their contents to standard output and sends out packets by reading them off the standard input. Similar to the relation of *sshd* and *inetd* (FreeBSD® programs), such generic adapter can connect a lot of services to Partov without changing their codes.

### D. Partov Topology Language

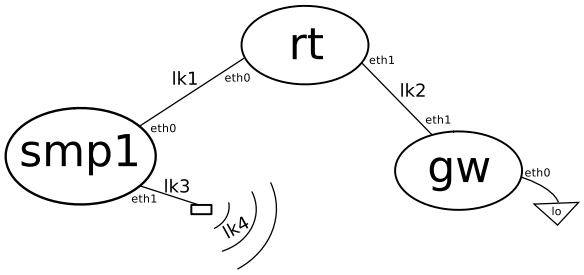
The *Partov Topology Language (PTL)* is an XML-based language for declaring topology descriptions in `.map` files. It defines the network's topology and how single nodes should communicate with each other. A sample topology is shown in Figure 4a and its corresponding PTL definitions are provided in Figure 4b, where a general example of map files including three mandatory parts (i.e. nodes, links, and lists) and two optional elements (i.e. authorization and FSM) are presented. Figure 4c expands on the nodes definitions of the example. These definitions are discussed in the rest of this section.

It should be noted that Partov supports two scheduling algorithms: Real-time and compact-time. With the compact-time approach, Partov moves the clock forward in order to finish simulation as fast as it could, depending on the processing power at hand. In the real-time mode, Partov executes the simulation in sync with real world events by additional wait/sleep calls. In this latter case, the Partov engine could connect the simulated toplogy to a real network and keep both synced time-wise. By default, Partov employs the real-time scheduler. However, when using the `realtime=false` option in the `map` tag (i.e. main tag of the topology description), assuming that no emulation is in use, compact-time scheduling will be enabled instead.

*1) Nodes:* The *nodes* element identifies network nodes which perform the core simulation like routing packets. These tags specify type of nodes (e.g. Wireless Access Point), assign some unique names to them, define their properties like network interfaces through the `interfaces` tag, and/or configurations through the `parameters` tag. Each node may operate in either of the following states: 1) As a basic node, like *smp1* from Figure 4c, replying to *ARP* and *ICMP ECHO* messages; used for testing functionality of more complicated nodes, or 2) As a plugin node, like *gw*, which can do from very basic simulations like connection bandwidth throttling to very complicated simulations like running a virtual application server.

For example the `gw` node, shown in Figure 4a and defined in Figure 4c, can be noted which is an *InternetGateWayNode* plugin. This plugin has two interfaces. Its first interface is connected to the loopback network device of hosting server and the second interface is connected to `lk2` virtual link. It also has two plugin specific parameters, the `param` subtags of the `parameters` tag. These parametric definitions in PTL allow any plugin to be easily configured via the standard PTL notation. Parameters can be used for any special configuration like configuring routing table of a *Router* plugin, *advertised networks* of an *InternetGateWayNode* plugin, and so on.

*2) Links:* The *links* element shapes the virtual topology. The links should be named so that it could be referenced, have a specified protocol (e.g. CSMA/CD or 802.3), and possibly annotated with link latency, packet loss probability, and maximum bandwidth. For example the third link in the Figure 4b is named `lk3` and is referenced by the forth link (viz., `lk4`) via its `connected-to-link` subtag. A link can be connected to other nodes' interfaces or links. Connecting links to nodes allows the node to send/receive packets over the link to/from other nodes. This is the common case in networks but consider the case that a node sends packets over a wireless channel. In addition to connecting two nodes together, a wireless channel can be modeled as a link where multiple




(a) The sample topology

```
<?xml version="1.0" encoding="UTF-8"?>

<ptl:map xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns:ptl="http://partov.ce.sharif.edu
    /2013/PTL/Map" xsi:schemaLocation="http://partov.ce.
    sharif.edu/2013/PTL/Map map.xsd" ptl:name="sample"
    ptl:version="3.1" ptl:realtime="true" ptl:count="12">
 <ptl:authorization ptl:resumable-by="instantiator" ptl:
     max-ipu="5">
  <ptl:instantiator>
   <ptl:group>lab1</ptl:group>
   <ptl:group>lab2</ptl:group>
  </ptl:instantiator>
  <ptl:connection-restriction>
   <ptl:host>127.0.0.1</ptl:host>
   <ptl:host>213.233.185.152</ptl:host>
  </ptl:connection-restriction>
 </ptl:authorization>
 <ptl:fsm ptl:initial-state="functional">....</ptl:fsm>
 <ptl:links>
  <ptl:repeat ptl:count="10">
   <ptl:link ptl:name="lk1" ptl:protocol="802.3"><ptl:
       bandwidth>10Mbps</ptl:bandwidth></ptl:link>
  </ptl:repeat>
  <ptl:link ptl:name="lk2" ptl:protocol="802.3"><ptl:
      bandwidth>10Mbps</ptl:bandwidth></ptl:link>
  <ptl:link ptl:name="lk3" ptl:protocol="csma-cd"><ptl:
      latency>45ms</ptl:latency></ptl:link>
  <ptl:link ptl:name="lk4" ptl:protocol="csma-cd" ptl:log
      ="true">
   <ptl:connected-to-link ptl:name="lk3" ptl:direction="
       inout"/>
   <ptl:loss>10%</ptl:loss>
  </ptl:link>
 </ptl:links>
 <ptl:nodes>....</ptl:nodes>
 <ptl:lists>
  <ptl:mac ptl:name="smp1-mac0">
   <ptl:item>00:24:8C:01:79:01</ptl:item>
   <ptl:item>00:24:8C:01:79:04</ptl:item>
  </ptl:mac>
  <ptl:ipv4 ptl:name="smp1-ip0">
   <ptl:item>192.168.121.1</ptl:item>
   <ptl:item>192.168.122.1</ptl:item>
  </ptl:ipv4>
  ....
 </ptl:lists>
</ptl:map>
```

(b) General structure of the PTL file

```
<ptl:nodes>
 <ptl:repeat ptl:count="10">
  <ptl:simple ptl:name="smp1">
   <ptl:interfaces>
    <ptl:ethernet-interface>
     <ptl:mac-address-ref>smp1-mac0</ptl:mac-address-ref>
     <ptl:ip-address-ref>smp1-ip0</ptl:ip-address-ref>
     <ptl:netmask>255.255.255.252</ptl:netmask>
     <ptl:delay><ptl:var>lat</ptl:var></ptl:delay>
     <ptl:connected-to-link ptl:name="lk1" ptl:running="
         true"/>
    </ptl:ethernet-interface>
    <ptl:ethernet-interface>
     <ptl:mac-address-ref>smp1-mac1</ptl:mac-address-ref>
     <ptl:ip-address-ref>smp1-ip1</ptl:ip-address-ref>
     <ptl:netmask>255.255.255.252</ptl:netmask>
     <ptl:connected-to-link ptl:name="lk3"/>
    </ptl:ethernet-interface>
   </ptl:interfaces>
  </ptl:simple>
 </ptl:repeat>
 <ptl:plugin ptl:name="rt"ptl:plugin-identifier="Router">
  <ptl:interfaces>
   <ptl:repeat ptl:count="10">
    <ptl:ethernet-interface>
     <ptl:mac-address-ref>rt-mac0</ptl:mac-address-ref>
     <ptl:ip-address-ref>rt-ip0</ptl:ip-address-ref>
     <ptl:netmask>255.255.255.0</ptl:netmask>
     <ptl:connected-to-link ptl:name="lk1" ptl:running="
         true"/>
    </ptl:ethernet-interface>
   </ptl:repeat>
   <ptl:ethernet-interface>
    <ptl:mac-address-ref>rt-mac1</ptl:mac-address-ref>
    <ptl:ip-address-ref>rt-ip1</ptl:ip-address-ref>
    <ptl:netmask>255.255.255.0</ptl:netmask>
    <ptl:connected-to-link ptl:name="lk2"/>
   </ptl:ethernet-interface>
  </ptl:interfaces>
 </ptl:plugin>
 <ptl:plugin ptl:name="gw" ptl:plugin-identifier="
     InternetGateWayNode">
  <ptl:interfaces>
   <ptl:physical-ethernet-interface ptl:device-name="lo">
    <ptl:mac-address-ref>gw-mac0</ptl:mac-address-ref>
    <ptl:ip-address-ref>gw-ip0</ptl:ip-address-ref>
    <ptl:netmask>255.255.255.0</ptl:netmask>
    <ptl:max-buffer-size>1MB</ptl:max-buffer-size>
   </ptl:physical-ethernet-interface>
   <ptl:ethernet-interface>
    <ptl:mac-address-ref>gw-mac1</ptl:mac-address-ref>
    <ptl:ip-address-ref>gw-ip1</ptl:ip-address-ref>
    <ptl:netmask>255.255.255.0</ptl:netmask>
    <ptl:connected-to-link ptl:name="lk2"/>
   </ptl:ethernet-interface>
  </ptl:interfaces>
  <ptl:parameters>
   <ptl:param ptl:name="internet-connection">
    <ptl:value>lo</ptl:value>
   </ptl:param>
   <ptl:param ptl:name="advertised-networks">
    <ptl:value-ref>gw-ip0</ptl:value-ref>
    <ptl:value-ref>ap-net</ptl:value-ref>
    <ptl:value>30</ptl:value>
   </ptl:param>
  </ptl:parameters>
 </ptl:plugin>
</ptl:nodes>
```

(c) Nodes definitions in PTL

Fig. 4. Sample topology and its corresponding PTL definitions

nodes are connected to it. Furthermore, connecting links to other links can be used for modeling the collision between distinct wireless receiving/transmitting areas. For more fine-grained control over wireless channels and/or their collision modeling, the direction of the link-to-link connections can be defined, making a link specially for sending and/or receiving. As an example the connection between `lk3` and `lk4` is a bidirectional connection, configured by `direction` parameter of the `connected-to-link` tag. The value of this parameter can be `in`, `out`, or `inout`.

Additionally, all packets, traversing through links, can be logged in the pcap (*TCPDUMP/LIBPCAP public repository* 2014) format log files by setting the `log` parameter to *true*. For example logging packets is enabled on `lk4` link as seen in Figure 4b. Saved pcap log files can be used later for offline analysis of the simulation with help of tools like tcpdump (*TCPDUMP/LIBPCAP public repository* 2014) or Wireshark (*Wireshark - Go deep.* 2013).

*3) Lists:* As noted at the beginning of this section, in response to requirement `R7`, a topology may need to be instantiated by multiple users in parallel. Hence a method is required for assigning distinguished resources like MAC addresses, IP addresses, and so on to different topologies instances. For example, in Figure 4c some values like netmask

of interfaces are defined directly, but some others like IP address of *smp1* simple node are defined indirectly using names, *smp1-ip0* in this case. These named values are called resources and obtain their values dynamically.

In Figure 4b, the `lists` tag contains one list subtag (e.g. `mac` or `ipv4` subtags) for each resource. Each resource is described by its name using the `name` parameter and its potential values shaped by the `item` subtags. Each topology instance has an instantiation index which can be used for assigning these resources values. So first element in a list will be given to the first instantiated virtual topology (i.e. the first arriving user), the second element to the second instance, and so on. For example in the first one, the first interface of node `smp1` will have `00:24:8C:01:79:01` and in the next instance, it will have `00:24:8C:01:79:04` as its MAC address. Indeed nodes and links create a template for virtual topologies, so assigning the defined resources to them, makes the topology ready to be instantiated and process network traffic.

*4) Authorization:* The CFs are identified within the NSS component using a username/password authentication mechanism. This mechanism is provided via the *User Management Interface (UMI)* at the frontend and supported via MySQL (*MySQL :: The world's most popular open source database* 2014) database as the backend. Each user, identified by a unique username, may instantiate new instances as specified by the `max-ipu` parameter (i.e. maximum instances per user) of a typical topology (see Figure 4b) or resume simulation of a previously instantiated virtual topology. By the `authorization` tag, it is possible to restrict users which are allowed to instantiate and/or resume simulation of a topology and/or connected host addresses.

### E. FSM

The *Finite State Machine (FSM)* can be used for creating dynamic topology structures. This feature can be used for many purposes, like modifying routing table entries of a virtual router or making nodes movable within a network. A lot of dynamic networking concepts can be modeled using CTMC (Ross, 2009, chap. 6). The PTL FSM provides a framework for using this model in simulations. For example consider the CTMC of an erroneous link shown in Figure 5a. This link may fail with an average rate of 0.1 failures per second and may return to a functioning state again with a rate of 0.7 repairs per second. This CTMC can be defined in PTL as shown in Figure 5b; defining a set of states (i.e. `state` subtags within the `fsm` tag) and their transitions (i.e. the `transitions` subtag).

If a transition, for example from *Functional* to *Failed* state is triggered, after an exponentially distributed random time, the state of the FSM will change and commands given through the `activity` subtag will be executed. The `activity` tag defines a list of commands to be executed when the FSM enters to the related state, in this example enabling/disabling the erroneous link. This allows one to model very complex dynamic state-dependent behaviors using the CTMC.

In addition, nodes, links, and interfaces can be placed within `repeat` tag to be cloned as many times as required. For example the `lk1` link and the `smp1` node are cloned 10 times as shown in Figure 4. Within the `repeat` tag, the `var` tag can be used to point to resources as defined by `lists`. So each cloned instance can have a different configuration. For example each cloned `smp1` node has a different delay value for its first interface, defined via *lat* time list.

In what follows, the Partov system will be observed within two viewpoints: the practical issues raising in an implementation and the provided performance and scalability limits.

## IV. Evaluation

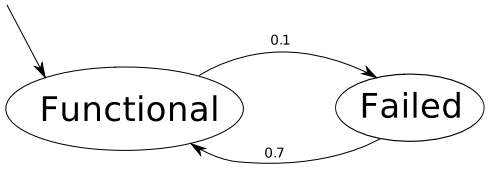
In this section, Partov's simulation/emulation capabilities are examined in terms of performance and scalability. To that end, the implementation details are discussed in Section IV-A, and then Partov's simulation capabilities are evaluated in Section IV-B and its emulation capabilities in Section IV-C.

### A. Implementation

The Partov system is implemented in C++ language using the Qt 4 libraries (*Qt Project* 2013). This brings two main benefits. First it makes Partov cross-platform. Second and more importantly, it provides a powerful signaling mechanism which makes network elements' communications more clear and well defined. The signaling mechanism enables event-driven simulation in which signals initiate and guide transmission of packets over links and through different nodes. These elements form object oriented networks (OON). Indeed each OON consists of three main element types: nodes, interfaces, and links. Each node may have multiple interfaces and each interface can be connected to multiple links. Currently links can run the *Carrier Sense Multiple Access (CSMA)* protocol for wireless connections and/or hub-based Ethernet networks, or the *802.3* protocol for switched networks with point-to-point links.

In support of multiple users, user management interface (UMI) makes it possible to define users and organize them in groups. Users can be authenticated via passwords and authorized for instantiating a virtual topology by defining the allowed instantiators groups. Each user is authorized to concurrently instantiate a specified number of topology instances, indicated through parameters in the map files. From each PTL file, description of a topology is extracted and from each topology, many instances can be instantiated for different users. Users can instantiate topologies, disconnect from the server and let the simulation to continue running on the server, and reconnect to them later or use the pcap log files to analyze the obtained results. In this way, users can even share some instantiated virtual topologies among themselves. Indeed, whenever a CF instance requests a connection to the Partov server, the *Simulation Server* component authenticates it and then either instantiates a new topology instance, and a new thread of execution, or locates a previously instantiated instance to be resumed and assigns it to the *CF*. Thus the CF receives packets from its remote agent and sends packets to it via the established SCFCP connection.

The well known *libpcap* (*TCPDUMP/LIBPCAP public repository* 2014) library is used for network emulation. In




(a) CTMC Model of an erroneous link

```
<ptl:fsm ptl:initial-state="functional">
 <ptl:state ptl:name="functional">
  <ptl:transitions>
   <ptl:transition ptl:rate="0.1" ptl:target="failed"/>
  </ptl:transitions>
  <ptl:activity>
   <ptl:do ptl:command="link-up" ptl:arg0="lk3"/>
  </ptl:activity>
 </ptl:state>
 <ptl:state ptl:name="failed">
  <ptl:transitions>
 <ptl:transition ptl:rate="0.7" ptl:target="functional"/>
  </ptl:transitions>
  <ptl:activity>
   <ptl:do ptl:command="link-down" ptl:arg0="lk3"/>
  </ptl:activity>
 </ptl:state>
</ptl:fsm>
```

(b) Corresponding FSM Definition of the designed CTMC

Fig. 5. Dynamic state-dependent behavior modeling

order to keep Partov cross-platform, it can be configured to use *winpcap* (*WinPcap - Home* 2013) on Windows™ platforms instead of *libpcap*. Another practical concern is memory usage. It's not efficient to copy frame objects, which may contain a kilobyte of data, when passing frames hop by hop. For solving this, the copy-on-write scheme (Accetta et al., 1986, Section 4) is used. This is implemented through a chain of proxy objects which can be copied very fast, serving a shared underlying chain of objects responsible for handling different network layers of packets.

And at last, the plugins which are created in response to the need of easy extensibility can be implemented as subclasses of the `PluginNode` class. Hence, new simulation models can be implemented in C++ using plugin infrastructure component facilities for generating, parsing, and managing packets or using timing facilities for accurate synchronized simulations. Plugins can access their unique-per-topology scheduler, configured by `realtime=true|false` parameter of the `map` tag in topology description, and schedule events in microseconds precision.

All evaluation scenarios, for acquiring each data point on the obtained plots, are exercised for 1 minute on an AMD Phenom™ Quad-Core Processor with $1.3GHz$ CPU and $4GB$ memory, running libpcap version 1.5.3 on Arch Linux 64-bit OS. Also each experiment is repeated several times and 95%-confidence interval of each plotted data point is found to ensure about statistical significance of the obtained results. Two used performance measures are average end-to-end delay of packets and their respective jitter values. Furthermore, two employed overhead measures are maximum resident set size (i.e. the amount of physical memory which is really being used) and average CPU utilization. These measures are collected using GNU time (*time - GNU Project - Free Software Foundation (FSF)* 1999) utility program. Here the jitter of each packet is defined by Eq. (1) as delay difference of consecutive packets. Packets' jitters of each flow are then aggregated according to Eq. (2) as defined in (Schulzrinne et al., 2003) to obtain their exponential moving average.

$$j_i = d_i - d_{i-1} \quad (1)$$

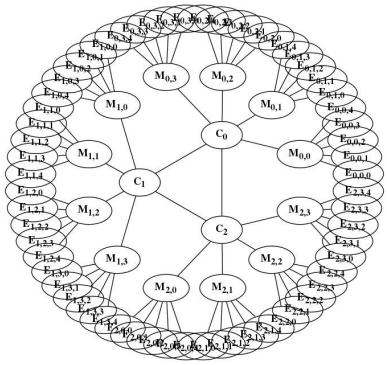


Fig. 6. Virtual Topology for Test of Scalability. There are 3 nodes at the core, each connected to 4 other nodes in the middle layer, and each is connected to 5 other nodes in the outer layer in turn.

$$\bar{j}_i = \frac{15}{16}\bar{j}_{i-1} + \frac{1}{16}|j_i| \quad (2)$$

In above formulas, $d_i$ is the delay, calculated for the $i$th packet, $j_i$ is the jitter, and $\bar{j}_i$ is flow's average jitter calculated till its $i$th packet. Having $n$ packets in a flow, the $\bar{j}_n$ is considered as the average jitter of the flow.

### B. Evaluating Simulation Capabilities

Each virtual topology consists of several nodes and links which can shape arbitrarily complex networks. In order to evaluate how well these topologies could be simulated and how scalable they are, two questions are investigated: 1) How many nodes can be simulated in each topology instance? and 2) How many packets can be processed within a specific time limit? The first dimension reflects on the scalability of topology indicating whether Partov can be used for simulating a CDN on a single PC, while the second dimension highlights its performance (i.e. maximum processing) limits.

To respond to the above noted questions, a series of topologies must be created and simulated, where these topologies should scale up in terms of virtual nodes, and in turn amount of generated traffic among them. One option is to use real ISP topologies which are made available for example by Rocketfuel (Spring, Mahajan, and Wetherall, 2002). But this does not allow the simulated topology to scale up gradually in

order to investigate how rapidly or smoothly the performance measures change. Another option is to generate a synthetic and parametric topology which can be instantiated in different sizes. Following the later option, a series of topologies like the one shown in Figure 6 are generated.

These topologies are made up of 3 layers and mimic different ISP tiers. The inner layer indicates the core network, the middle layer corresponds to the edge network, and the outer layer corresponds to where end-hosts reside. Clients and servers are scattered uniformly in the outer layer of Figure 6. It's possible to generate a topology knowing desired nodes' fan-out degrees and number of clients/servers pairs. Each client selects a server, the farthest server in this evaluation in order to check the worst case, pings it periodically, receives its response, and calculates the round-trip delay.

A notable feature in the IEEE 802.3 protocol is its ability to work in full-duplex mode (Frazier and Johnson, 1999) with point-to-point links and put packets in the pipeline which can affect the congestion point in simulations. All links are configured to run 802.3 protocol, their latencies are set to $1ms$ and each interface's buffer size is set to $500B$. The traffic sending rate can be increased by reducing the gap interval at which new ICMP packets are sent out in a topology having 7 core routers with fan-out degree of 14, which in turn are connected to edge routers with fan-out degree of 49. This leads to a large topology with $7*(14*(49+1)+1) = 4907$ nodes and $7 * (14 * (49 + 1)) + \frac{6*7}{2} = 4921$ virtual point-to-point links. In each topology instance, the paths between clients/servers pairs consist of 10 hops, 5 links to reach the server and 5 to return to the client. So simulating in real-time, it's expected to observe an overall RTT of $10ms$.

At the beginning, new ICMP Echo requests are generated once per $100ms$ and this gap is reduced gradually. Enabling 30 clients/servers pairs in this topology and reducing inter-batch gap intervals from $100ms$ to $10ms$, causes sending rate to increase from $300pps$ till $3000pps$. Results of this experiment are shown in Figure 7. Testing different scenarios, physical memory usage remained between $106.3MB$ and $106.4MB$ and lengths of calculated 95%-confidence intervals for expected delay values remained smaller than $0.05ms$ which is a sign of highly repeatable simulations.

Sending packets with rate of $r$, would mean that a new batch of packets will be sent out once per $\tau = 1/r$ by 30 clients towards 30 servers. If all of these packets could reach their destination and their responses are received in less than $\tau$, then when the next batch of packets are sent out, no packets from the previous batch will be in the network and there will be no competition for the network resources among the two batches. But if those packets require more time, some packets will remain in the network (i.e. in router buffers or on the links) while the new batch of packets enters the network. So number of packets which must compete for resources becomes more, and more packets will remain in the network at the end of next $\tau$ period, and so on. After a short time the network will be congested with a lot of packets, each incurring a lot of delay, shaping a jump in the delay curve. This theoretical delay limit, before the congestion point, is drawn with solid line in Figure 7a, and measured average delays of experiments are drawn with a dotted line. The simulator measures low delay values and scales well with packets generation rate till congestion point and shows a jump in delays after that point. Jitter values are also limited to $2.5ms$ as indicated in Figure 7b. Minimum delay, considering the designed topology, is $10ms$ and the difference with this minimum delay is a measure of simulator's performance. Correspondingly, increase of rate, increases packets which simulator has to process in real-time and so increases the CPU usage as shown in Figure 7c.

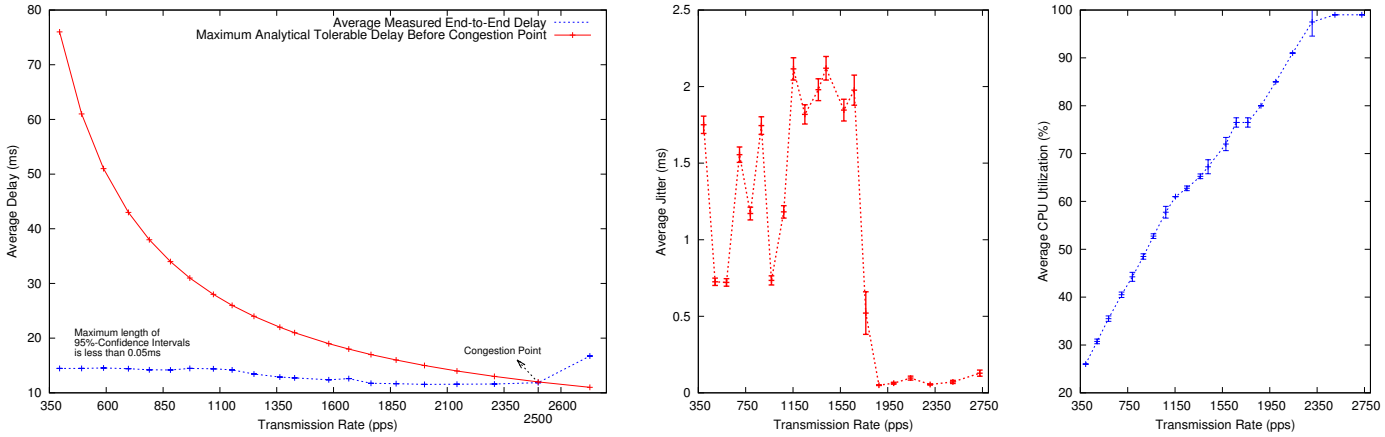
The congestion point in this experiment occurs at rate of $2500pps$. The exact achievable real-time packet processing rate depends both on the simulator and the simulated network. For example in above experiment, congestion occurs when delay reaches $11.9ms$, while a rate of $2500pps$ is generated by 30 servers sending packets batches once per $12ms$.

To test the performance limits in a more difficult situation, in another experiment the number of servers is increased, each sending with a rate of $10pps$, to increase both sending rate and topology size simultaneously. Figure 8 shows results of this experiment. Number of active clients/servers pairs is increased from 4 to 564, each scenario is exercised for 1 minute and repeated several times and delay samples are collected. In all cases, lengths of calculated 95%-confidence intervals for expected delay values were less than $1ms$. Also the maximum resident set size for all cases was between $103MB$ and $107MB$. Increased memory usage is due to additional virtual nodes, each one adding $500B$ of buffered packets and also adding to the total delay as packets can wait more in queues.

As seen in Figure 8a, increase in traffic amount and delay values are tied together linearly which is a sign of good scalability. Figure 8a for delay is constructed from two parts separated by a significant jump, indicating the congestion point. Points belonging to pre-congestion point, are scaled in center of figure to be seen more clearly. Furthermore the jitter shown in Figure 8b is below $1ms$. Moreover, and as illustrated in Figure 8c, CPU utilization increases linearly too and near the congestion point one core is utilized completely. In this experiment, $5240pps$ were being processed before the congestion point. It should be noted that $5240pps$, just like the $2500pps$ from previous experiment, does not mean that simulator itself has an upper limit. But this value is dependent on the simulation scenario as well as the simulator hardware.

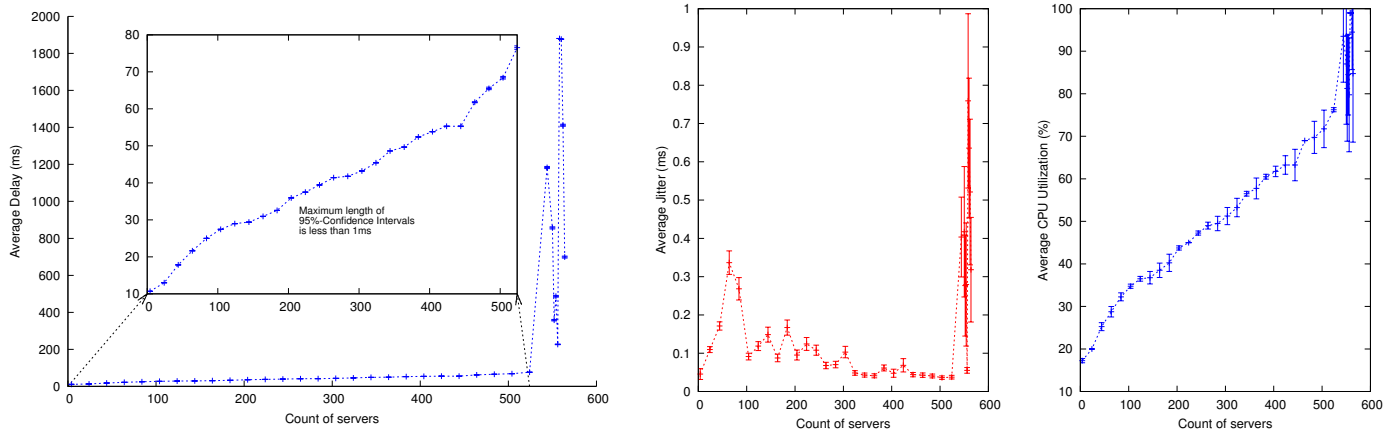
Lastly, the non real-time scheduler supported by Partov was evaluated. This scheduler is usable when no emulation is in the circuit, to either reduce the wall-clock by removing sleeps when possible or increase it in favor of more accuracy when the simulator hardware can not keep up with the simulated topology. In this experiment exactly the same conditions of previous simulation are set, else of the scheduler which is configured to work in compact-time mode. Memory usage remained between $103MB$ and $106MB$ for all cases. Here, CPU is utilized completely and wall-clock is the main metric indicating overhead of simulation.

Figure 9 shows that the wall-clock time starts from $5.8s$ and increases linearly to $128.5s$ where $604$ servers are sending packets over the described topology. For example, compact-time simulation requires about $60s$ for a 60-seconds simulation




(a) Average packet delay (b) Average packet jitter (c) Average CPU utilization

Fig. 7. Average performance in respect to traffic sending rate. Simulated topologies have 7, 98, and 4802 nodes respectively in core, middle, and outer layers and latency of all links are set to $1ms$.




(a) Average packet delay (b) Average packet jitter (c) Average CPU utilization

Fig. 8. Average performance in respect to the servers' count. Simulated topologies have 7, 98, and 4802 nodes respectively in core, middle, and outer layers and latency of all links are set to $1ms$. Pre-congestion part of the delay plot is zoomed in and shown at the center.

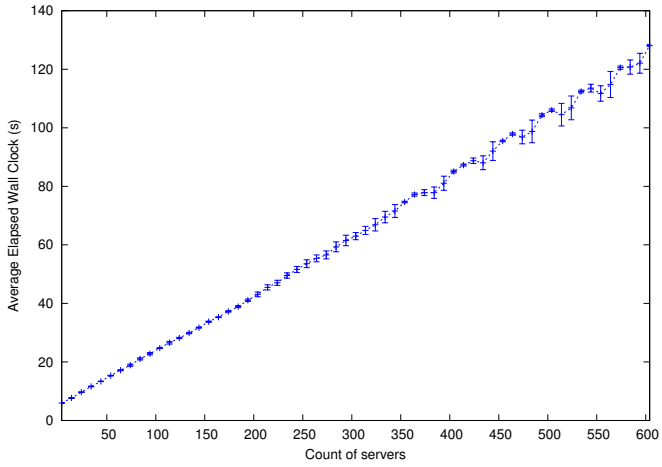



Fig. 9. Average elapsed wall-clock to complete 60 seconds of simulation using compact-time scheduling. Simulated topologies have 7, 98, and 4802 nodes respectively in core, middle, and outer layers and latency of all links are set to $1ms$.

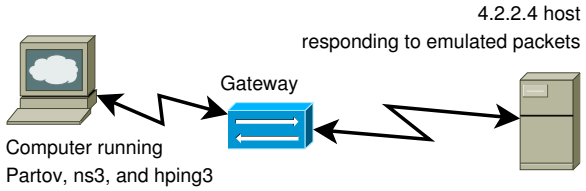
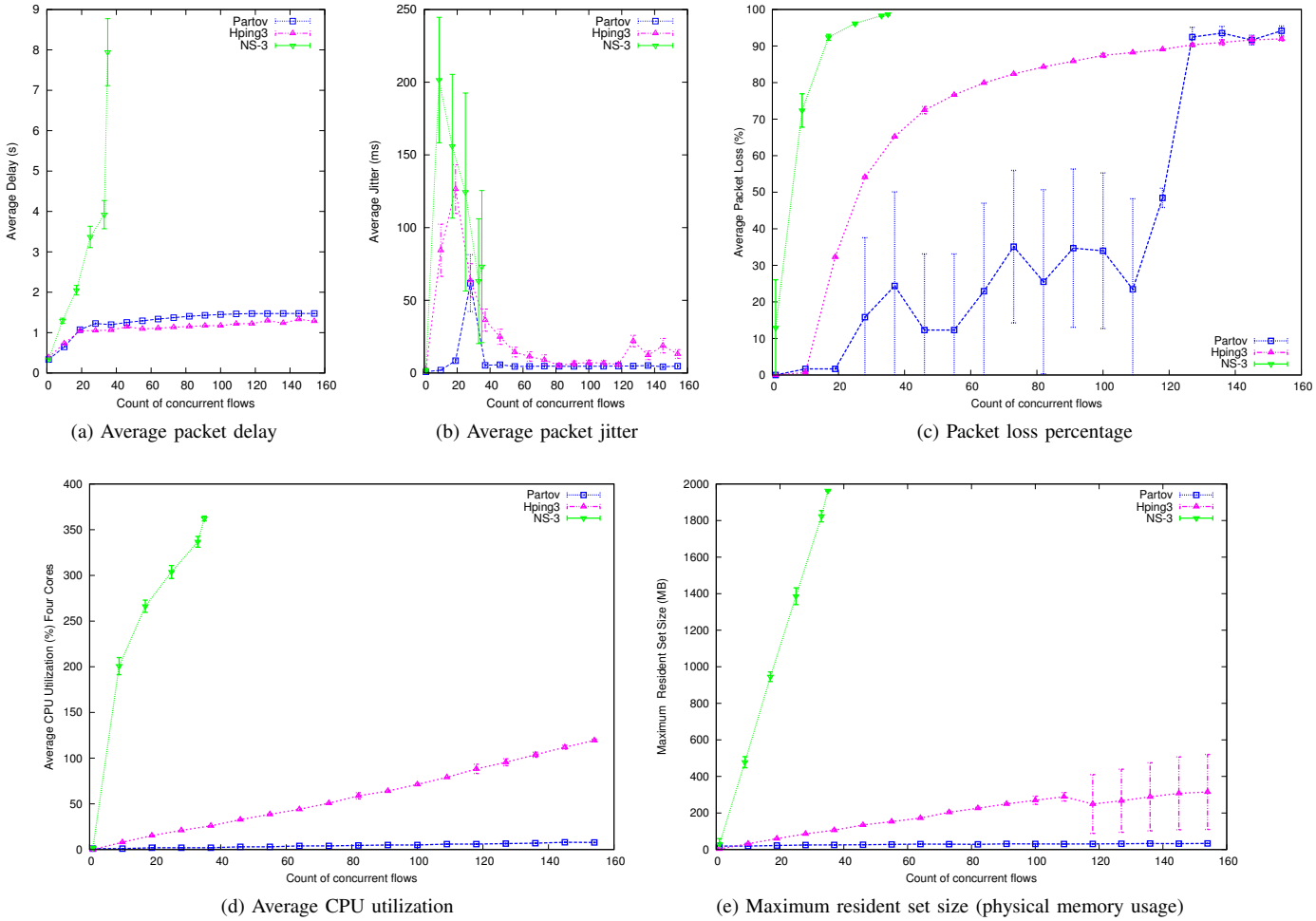



Fig. 10. The used physical topology in emulation experiments

when about 300 servers are generating traffic with a rate about $3000pps$. Furthermore, after finishing each 60-seconds of simulation, delay and jitter values are calculated with zero error in all cases.

### C. Evaluating Emulation Capabilities

This section evaluates the Partov system's emulation performance through comparing it with performance of the *Hping3* (*Hping security tool - Hping3 information* 2006), the well known packet generator tool, and performance of the *NS-*




(a) Average packet delay (b) Average packet jitter (c) Packet loss percentage

(d) Average CPU utilization (e) Maximum resident set size (physical memory usage)

Fig. 11. Average performance in respect to the number of emulated concurrent flows in Partov, NS-3, and Hping3

*3*[1] (Henderson, 2010), one of the most famous and widely used simulators. For this purpose, the physical topology shown in Figure 10 has been employed. That testbed consists of a physical computer, acting as the client and serving emulators, connected to a router (i.e. gateway) through an initially idle Ethernet link which is connected through Internet to the 4.2.2.4 host, acting as the server and responding to emulated packets.

Partov starts emulating a real node, sending ICMP packets with $1024B$ payload in each ICMP packet with rate of $1pps$ to the real node. Packets traverse that physical path, return to the virtual world, and get used for calculating delay and jitter samples. This can be compared with an Hping3 process pinging 4.2.2.4 with same rate and packet length. The NS-3 is configured similarly to have just one emulated node, working in real-time. As Hping3 is a real process, dedicated to packet generation tasks, its delay/jitter values can be considered as a worthy base-line of emulation accuracy; the more similar to a real process traffic patterns, the more accurate emulation. This experiment is repeated several times and 95%-confidence interval of average delay/jitter values are calculated. The 95%-confidence interval of expected delay value of all three tools is found to be within $365.6 \pm 5.9ms$ interval. This shows that all tools are similarly accurate in the basic single-node scenario.

This scenario can be scaled up by instantiating more topology instances in Partov and NS-3 and running more concurrent Hping3 processes. Considering average delay and jitter values as shown in Figure 11a and Figure 11b respectively, Partov's measured delay values follow Hping3's measurements closely. Scaling number of emulated concurrent flows upto 154, Partov's error in delay measurements, relative to Hping3's, remain less than $275ms$. Also its jitter is less than Hping3's which means that packets are sent/received more regularly. It also suffers from lower loss rate as visible in Figure 11c where loss percentage of Partov's packets is less than Hping3's upto 127 concurrent flows, and approximately equal to it thereafter.

NS-3 has severe problems in terms of scalability. Its CPU utilization hits 362% (i.e. four CPU cores are almost completely utilized) by 35 emulated concurrent flows as seen in Figure 11d, its measured delay reaches $7.9s$, more than 6.8 seconds error relative to Hping3's, and its loss percentage reaches 98%. Also its memory utilization is much more than the other two tools (see Figure 11e) as it fills $2GB$ while Hping3 is using $315MB$ and Partov is using just $34MB$ of physical memory.

[1]NS 3.19 is used in the experiments.

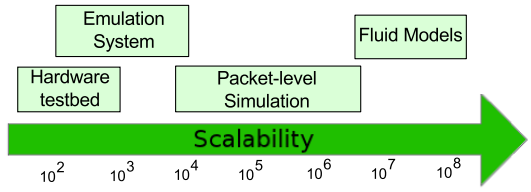



Fig. 12. Solutions scalability as nodes count (based on (Reineck, 2008))

Scaling each tool until the point that loss percentage crosses $90\%$ limit, it can be concluded that high CPU usage prevents NS-3 from scaling beyond one fourth of where Partov and Hping3 can scale. Also if a more powerful CPU was being used, the rapid rate of memory usage would made physical memory the next bottleneck of NS-3. It must be noted that as different emulated topology instances are considered to be independent, the ideal scenario is the one that even scaling and emulating hundreds of topology instances, each instance still performs like one single Hping3 process. However, the ideal case can not happen due to processing power limitations. This causes multiple Hping3 processes to show more delay, slowing each other. However the more light weight implementation of Partov allows it to resist more upon this non-desired delay increasing phenomena.

## V. Related Work

There is a spectrum of tools starting from simulation-based solutions like *Network Simulator (NS-2)* (*The Network Simulator - NS-2* 2011) to hardware-laden solutions like *PlanetLab* (*PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services* 2012). The former minimizes the required time for acquiring experiments' results while the later maximizes the precision of the results. However, it's possible to balance this equation and achieve easy and fast experiments while staying at a desired accuracy level using the middle-way solutions like *NCTUns* (Wang and Huang, 2012), *IMUNES* (Zec and Mikuc, 2004), *Network Simulator (NS-3)* (Henderson, 2010), *OMNeT++* (Varga and Hornig, 2008), or *Virtual Network System (VNS)* (Casado and McKeown, 2005). By moving in this spectrum towards simulation-based solutions, more scalable tools will be available. In terms of scalability four categories can be found (Reineck, 2008) as shown in Figure 12. This section describes this spectrum of tools as compared to Partov.

At one extreme, the *PlanetLab* (*PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services* 2012) from the hardware-laden category, uses physical computers distributed all around the world which is specially useful for testing p2p applications. The *Emulab* (*Emulab - Network Emulation Testbed Home* 2014) alleviates this situation a little via a software layer above a smaller hardware testbed virtualizing computers, routers, and switches, emulating the Internet in a small scale to perform experiments. Anyway, in this category, count of available devices limits the scalability of built network topologies.

Other solutions sit in the *Packet-level* simulation category (OPNET in one mode can use *Fluid Models* too). However, middle-way solutions (NCTUns, IMUNES, OMNeT++, NSE, NS-3, VNS, and Partov) combine *Packet-level* simulation and *Emulation* categories to do emulation while interacting with other physical network nodes and switch back to *Packet-level* simulation for the rest. The *NS-2* (*The Network Simulator - NS-2* 2011) software which is one of the most famous tools (*WNS2 '06: Proceeding from the 2006 workshop on NS-2: the IP network simulator* 2006), concentrates on pure simulations. It allows topology creation using an imperative language, called *OTcl*, which makes topology description another programming task itself, while simulation models like nodes and interfaces are implemented in C++ modules of the NS-2. This solution is extended as *NSE* (Nethi et al., 2007) with network emulation abilities for networked control systems (NCS). This extension provides a *tap agent* which can read/write packets from/to the live network. Its newer version, the *NS-3* (Henderson, 2010), adds emulation capability and drops *OTcl* completely, requiring topologies to be coded in C++ or Python.

The NCTUns (Wang and Huang, 2012) and IMUNES (Zec and Mikuc, 2004) solutions employ kernel facilities in favor of more performance. In the NCTUns (Wang and Huang, 2012), kernel-reentering is used to allow the Linux TCP/IP protocol stack to be reused in the emulated network. IP addresses in the emulated network are limited to the "1.0.subnet.host" form. This limitation is due to the used emulation method in NCTUns. For routing packets, the Linux kernel routing tables are being used. For avoiding route conflicts, as all routes are inserted in one shared routing table, when a node with address "1.0.a.b" sends a packet towards a node with address "1.0.c.d", the route is formed as "a.b.c.d". This requires those two bytes of IP address to make it fit in normal Linux route definitions. IMUNES (Zec and Mikuc, 2004) removes this limitation by replicating network stack, private state variables, routing tables, etc. in the FreeBSD kernel. In IMUNES, each virtual node contains a complete replica of above items and is connected to other virtual nodes via kernel space links.

The *OPNET* (Kaparti, 2005), *OMNeT++* (Varga and Hornig, 2008), and *VNS* (Casado and McKeown, 2005) solutions provide simulation capabilities too, but they also recognize the insufficiency of pure simulation, trying to incorporate with real network in some manner. The *OPNET* (Kaparti, 2005) can use packet-level simulation –while doing discrete event simulation (DES)– or employ flow-based analysis through fluid models. However, it misses interaction with live network and limits interaction with real network to captured traffic files. By resending the captured traffic, it can emulate existence of a real application in the virtual network. However this ignores all effects of virtual network traffic on the application traffic, which is captured and so is fixed, and vice versa.

The *OMNeT++* (Varga and Hornig, 2008) tries to divide the problem more coherently. Typically 3 parts get involved in this process. First, the simulation active behavior is coded in C++. Then these codes (i.e. *simple modules*) will be combined to shape larger *compound modules* in a declarative language (i.e. *NED*). These two parts describe a complete topology. Third part, written in *ini* files, configure parameters for each experiment. The OMNeT++ uses packet-level simulation too.

TABLE I
FEATURES OF COMPETENT SOLUTIONS. THE ∼, ✓, AND × INDICATE THE ALTERNATIVE APPROACH (SEMI-SUPPORTED), SUPPORTED, AND NON-SUPPORTED FEATURES. THE - MEANS THAT FEATURE IS NOT APPLICABLE TO THAT SOLUTION.

| Row | Covered Requirement | Features | PlanetLab | Emulab | NCTUns | IMUNES | VNS | Partov | NS-3 | NSE | OMNeT++ | OPNET | NS-2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | R0 | *Virtual network simulation* | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 2 | R1 | *Emulation/Real network interaction* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ∼ | × |
| 3 | | *Real clock synchronization* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| 4 | R2 | *Out of the box simulation models* | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ |
| 5 | R3 | *General purpose tool* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ | ✓ | ✓ |
| 6 | | *Extensible simulation platform* | × | × | × | × | × | ✓ | × | × | ✓ | × | × |
| 7 | R4 | *Declarative network description* | ∼ | ∼ | ∼ | ∼ | ✓ | ✓ | × | × | ✓ | ∼ | × |
| 8 | R5 | *Dynamic topology change* | × | × | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | × | ✓ |
| 9 | | *Markovian network modeling* | × | × | × | × | × | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| 10 | R6 | *Distributed deployment* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | × | × |
| 11 | R7 | *Support for multiuser simulation* | ✓ | ✓ | ✓ | × | ✓ | ✓ | × | × | × | × | × |
| 12 | R8 | *Multi-thread simulation* | - | - | ✓ | ✓ | ✓ | ✓ | ✓ | × | × | ✓ | × |

Nevertheless, it does not provide any simulation logic like routers, etc. by itself. OMNeT++ just provides the simulation core and leaves the rest to be implemented independently by other frameworks. This makes OMNeT++ very abstract. It should be noted that there are some frameworks for normal network simulation, mobility, and so on. It's also possible to support network emulation via frameworks.

The *VNS* (Casado and McKeown, 2005) software provides both emulation capability, out of the box, and distributed simulation using a client/server deployment architecture. It focuses on the first three layers of the OSI network reference model (*Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model* 1996). The VNS server contains the simulation core which can route incoming packets, both simulated and emulated ones, in the virtual network. Some nodes in the simulated network can be marked for remote controlling. Packets received by those nodes will be sent to the VNS stubs (i.e. clients) to be processed and replied back. This can be useful to incorporate the real network conditions (e.g. latency) in simulations specially if a physical testbed is available.

The above noted solutions are compared in Table I based on 12 features and how they compare to the requirements noted in Section II. According to this table, hardware-laden solutions are not promising else of emulation and physical network related features. They also suffer from high initial deployment costs. On the other hand, NS-2 as a pure simulation solution is easily deployable but lacks emulation and extensibility to support new areas. NSE fixes the emulation support for networked control systems (NCS), but still it's too specialized and there is no supported solution for introducing new modules, else of changing the NS-2 base itself. OPNET is limited to repeating offline packet capture files in the simulated network which does not provide any considerable interactivity outside the virtual world. NCTUns and IMUNES have limited simulation capabilities in comparison to others (i.e. more concentrated upon emulation). It can be concluded that NS-3, OMNeT++, VNS, and Partov perform better than others in terms of features support. As shown in Section IV, Partov system is easily scalable both in simulation, scaling up linearly with frame processing rate, and in emulation, easily following performance of the *Hping3*.

Another aspect is the method of network topology description. Two general language types exist: imperative and declarative. This is done with OTCL in NS-2, C++ or Python codes in NS-3, NED in OMNeT++, and PTL in Partov. NSE is similar to NS-2 in this aspect. OPNET, NCTUns, and IMUNES provide GUI instead of textual topology descriptions which is an alternative declarative topology model, despite imperative descriptions provided in programming languages (C++, Python) and OTcl. One benefit of declarative languages is their machine readability. It is not so hard to read and convert a declarative topology description from one format to another like what FNSS (Saino, Cocora, and Pavlou, 2013) tool does or provide a graphical representation for hand-written topologies.

Another investigated feature is extensibility and how a solution answers to the need of new modules. As no solution may contain all possible modules out of the box, it's important to have a method for adding modules as needed. OPNET approaches this issue by supporting a lot of simulation modules. But it's the hardest tool among others if a new module become required. On the other hand, OMNeT++ stands on an extreme and leaves implementation of all simulation modules to separate frameworks. Partov system both provides basic simulation models readily and provides easy extensibility via plugins. Also it's useful to make simulation configurations out of the simulation code, like PTL in Partov or ini and NED files in OMNeT++. This allows the same simulation logic to be connected to other modules like LEGO parts which is more handy in network design.

## VI. Conclusion

In this paper the Partov is presented, an extensible and reliable simulation/emulation engine. The architecture of Partov, makes it possible to extend simulation modules easily through the concept of plugins. Also the Partov is compared with some other simulations tools and showed that Partov can be used for large scale simulations. The main advantages of using Partov system can be summarized as

1) coordinating simulation and emulation capabilities in a coherent manner,
2) providing a rich and extensible simulation framework,
3) providing a concrete set of simulation models for out-of-the-box usage,
4) being scalable and reliable,
5) providing extensive logging capabilities for detailed analysis.

Also, it should be noted that Partov is currently available as an open source tool, under GPLv3, and can be downloaded from <http://partov.ce.sharif.edu/gitlab/public>. It has also been employed, for the past few years, as part of the computer networks course (Momeni and Kharrazi, 2012) being taught at Sharif University of Technology.

## References


Accetta, Michael J. et al. (1986). "Mach: A New Kernel Foundation for UNIX Development". In: *USENIX Summer*, pp. 93–113.

Arlitt, Martin F and Carey L Williamson (1996). "Web server workload characterization: The search for invariants". In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 24. 1. ACM, pp. 126–137.

Casado, Martin and Nick McKeown (2005). "The virtual network system". In: *SIGCSE Bull.* 37.1, pp. 76–80. ISSN: 0097-8418. DOI: http://doi.acm.org/10.1145/1047124.1047383.

*Emulab - Network Emulation Testbed Home* (2014). URL: http://www.emulab.net/ (visited on 02/12/2014).

Frazier, H. and H. Johnson (1999). "Gigabit ethernet: From 100 to 1,000 mbps". In: *Internet Computing, IEEE* 3.1, pp. 24–31.

Henderson, Tom (2010). "NS-3 Tutorial". In: *9th GENI Engineering Conference (GEC9) Workshops and Tutorials*.

*Hping security tool - Hping3 information* (2006). URL: http://www.hping.org/hping3.html (visited on 02/12/2014).

*Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model* (1996). International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC). URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip.

Kaparti, Ranjan (2005). "OPNET IT Guru: A Tool for Networking Education". MSCIT Practium Paper. Regis University.

Lisha, G. and L. Junzhou (2006). "Performance analysis of a p2p-based voip software". In: *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*. IEEE, pp. 11–11.

Momeni, Behnam and Mehdi Kharrazi (2012). "Improving a Computer Networks Course Using the Partov Simulation Engine". In: *Education, IEEE Transactions on* 55.3, pp. 436–443. ISSN: 0018-9359. DOI: 10.1109/TE.2012.2183597.

*MySQL :: The world's most popular open source database* (2014). URL: http://www.mysql.com/ (visited on 02/12/2014).

Nethi, Shekar et al. (2007). "Platform for emulating networked control systems in laboratory environments". In: *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*. IEEE, pp. 1–8. DOI: 10.1109/WOWMOM.2007.4351727.

Park, C. et al. (2006). "Capturing the elusive poissonity in web traffic". In: *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*. IEEE, pp. 189–196.

*PlanetLab — An open platform for developing, deploying, and accessing planetary-scale services* (2012). URL: http://www.planet-lab.org/ (visited on 02/12/2014).

*Qt Project* (2013). URL: http://qt-project.org/ (visited on 02/12/2014).

Reineck, Karsten M. (2008). "Evaluation and Comparison of Network Simulation Tools". MA thesis. University of Applied Sciences, Bonn-Rhein-Sieg, Department of Computer Science.

Ross, Sheldon M (2009). *Introduction to probability models*. Academic press.

Saino, Lorenzo, Cosmin Cocora, and George Pavlou (2013). "A toolchain for simplifying network simulation setup". In: *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques, ser. SIMUTOOLS*. Vol. 13.

Schulzrinne, H. et al. (2003). *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550 (INTERNET STANDARD). Updated by RFCs 5506, 5761, 6051, 6222. Internet Engineering Task Force. URL: http://www.ietf.org/rfc/rfc3550.txt (visited on 02/12/2014).

Spring, Neil, Ratul Mahajan, and David Wetherall (2002). "Measuring ISP topologies with Rocketfuel". In: *ACM SIGCOMM Computer Communication Review* 32.4, pp. 133–145.

*TCPDUMP/LIBPCAP public repository* (2014). URL: http://www.tcpdump.org/ (visited on 02/12/2014).

*The Network Simulator - NS-2* (2011). URL: http://www.isi.edu/nsnam/ns/ (visited on 02/12/2014).

*time - GNU Project - Free Software Foundation (FSF)* (1999). URL: http://www.gnu.org/software/time/ (visited on 02/18/2014).

Varga, András and Rudolf Hornig (2008). "An overview of the OMNeT++ simulation environment". In: *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. Marseille, France: ICST (Institute for

Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 1–10. ISBN: 978-963-9799-20-2.

Wang, Shie-Yuan and Yu-Ming Huang (2012). "NCTUNS DISTRIBUTED NETWORK EMULATOR". In: *Internet Journal* 4.2, pp. 61–94.

*WinPcap - Home* (2013). URL: http://www.winpcap.org/ (visited on 02/12/2014).

*Wireshark - Go deep.* (2013). URL: http://www.wireshark.org/ (visited on 02/12/2014).

*WNS2 '06: Proceeding from the 2006 workshop on NS-2: the IP network simulator* (2006). Pisa, Italy: ACM. ISBN: 1-59593-508-8.

Zec, Marko and Miljenko Mikuc (2004). "Operating system support for integrated network emulation in imunes". In: *Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, pp. 3–12.