A Simulation Suite for Accurate Modeling of IPv6 Protocols

Johnny Lai[†] Eric Wu[†] Andràs Varga[‡] Y. Ahmet Şekercioğlu[†] Gregory K. Egan[†]

[†]*Centre for Telecommunication and Information Engineering, Monash University, Melbourne, Australia*
[‡]*Department of Telecommunications, Technical University of Budapest, Hungary*

Abstract

As part of our ongoing research program on performance analysis of protocols for mobility management in IPv6 networks, we have developed a set of OMNeT++ models for accurate simulation of IPv6 protocols. Our simulation set models the functionality of the RFC 2373 *IP Version 6 Addressing Architecture* [5], RFC 2460 *Internet Protocol, Version 6 (IPv6) Specification* [3], RFC 2461 *Neighbor Discovery for IP Version 6 (IPv6)* [7], RFC 2462 IPv6 *Stateless Address Autoconfiguration* [10], RFC 2463 *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification* [2], and RFC 2472 *IP Version 6 over PPP* [4].

1 Introduction

Inevitably, telecommunication networks are increasingly becoming more complex as the trend towards the integration of telephony and data networks into integrated services networks gains momentum. It is expected that these integrated services networks will include wireless and mobile environments as well as wired ones. As a consequence of the rapid development and fusion of communication technologies, understanding the dynamic interaction of protocols and performance analysis are becoming much more complex to be investigated in small-scale experimental testbeds. Analytical analysis is also not feasible for similar reasons. Simulation is now considered as a tool of equal importance (as complementary to the analytical and experimental studies) for investigating and understanding the behavior of complex systems.

As part of our ongoing research programs on analysis of protocol performance on mobile IPv6 networks, we have developed a set of OMNeT++ models for accurate simulation of IPv6 protocols. We have chosen OMNeT++ as the simulation framework because of the following reasons: (a) It allows the design of modular simulation models, which can be combined and reused flexibly; (b) It is possible to compose models with any granular hierarchy; (c) OMNeT++ is open-source, free for non-profit use, and has a fairly large and active user community; (d) It has support for parallel simulation; and (e) Its performance is comparable to commercial simulation tools. Section 2 presents a brief summary of the features of OMNeT++.

Our IPv6 simulation model suite consists of several functional blocks. There is also dual-stack support for analysis of protocol interactions in mixed IPv4-IPv6 networking environments. The accuracy of the simulation is ensured because of the fine-grained level of details in the simulation. Realistically formatted protocol data units (PDUs) are passed between simulated network entities and service data units (SDUs) exchanged between the adjacent protocol layers. The IPv6 datagram format currently includes most of the extension headers except the ones related to Authentication and Encapsulating Security Payload. Real data from our network testbed is used to calibrate the model, and simulated processing delays are introduced where necessary to account for the differences without sacrificing performance. The structural breakdown of the model and module descriptions can be found in Section 3.

Currently we are working on mobility support, and future enhancements in the pipeline include profiling the model for very large scale network simulations (i.e. 10,000+ network entities) and dynamic creation of network topologies through XML (extensible markup language) based configuration files.

2 OMNeT++ Simulation Framework

OMNeT++ is a C++-based discrete event simulation package developed at the Technical University of Budapest by András Varga [8, 12]. The primary application area of OMNeT++ is the simulation of computer networks and other distributed systems. It is open-source, free for non-profit use, and has a fairly large and active user community. It also allows the design of modular simulation models, which can be combined and reused flexibly. Additionally, OMNeT++ allows the composition of models with any granular hierarchy. It has been shown that this simulation framework is suitable for simulation of complex systems like Internet nodes and dynamics of TCP/IP protocols realistically [6, 14].

Simulated models are composed of hierarchically nested modules. In OMNeT++, there are two types of modules: simple and compound modules. Simple modules form the lowest hierarchy level and implement the activity of a module, and they can arbitrarily be combined to form compound modules. Modules communicate with message passing. Messages can be sent either through connections that span between modules, or directly to their destination modules. The user defines the structure of the model (the modules and their interconnection) by using the topology description language (NED) of OMNeT++ [11].

Simple modules are implemented in C++, using the simulation kernel system calls and the simulation class library. For each simple module, it is possible to choose between process-style and protocol-style (state machine) modeling. Therefore, different parts of computing and communication systems can be programmed in their natural way and connected easily. The simulation class library provides a well-defined application programmer's interface (API) to the most common simulation tasks, including: random number generation; queues, arrays and other containers; messages; topology exploration and routing; module creation and destruction; dynamic topologies; statistics; density estimation (including histograms, P2 and k-split [13]); output data recording. The object-oriented approach allows the flexible extension of the base classes provided in the simulation kernel.

Model components are compiled and linked with the simulation library, and one of the user interface libraries to form an executable program. One user interface library is optimized for command-line and batch-oriented execution, while the other employs a graphical user interface (GUI) that can be used to trace and debug the simulation (as an example, Figure 1 shows a simulated network configuration). The GUI makes the internals of a simulation model fully visible: it displays the network graphics, animates the message flow and lets the user peek into objects (messages, queues, etc.) within the model. It is also possible to change parameters and message fields for debugging purposes. Visualization features make OMNeT++ suitable also for educational or demonstration purposes. Because of the modular design, it is possible to embed the simulation engine (including models) into other applications. OMNeT++ also has support for parallel discrete event simulations (PDES).

3 IPv6 Simulation Model

The IPv6 simulation model suite consists of several functional blocks. As one can expect, the major blocks reside in the network and data link control layers. These blocks can be connected together to form simulated hosts, routers, Ethernet hubs, point-to-point links etc. Figure 2 shows these blocks in a model of a router with three network interfaces. The core module (`IPProcessing`) of the network layer (Figure 2(b)) provides dual-stack support (IPv4 and IPv6).

Our simulation model provides enhancements to the existing OMNeT++ IPv4 models mainly in the areas of providing interchangeable network interfaces for simulating IP protocols using various physical transport mechanisms (point-to-point links, Ethernet connections etc.). The enhancements also include the ability to model nodes having any combination of these physical devices.

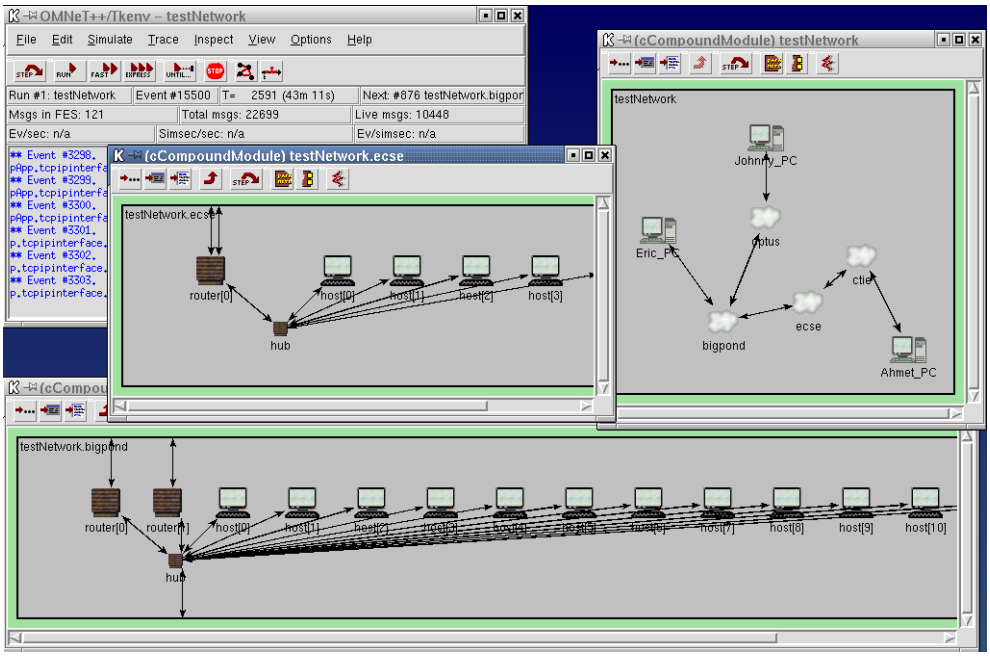


Figure 1: OMNeT++ screen showing a hierarchical network model. The upper right window shows the topology of a simulated network consisting of four subnets. Topologies of the two of these subnets, `ecse` and `bigpond` are shown in the middle and bottom windows respectively.

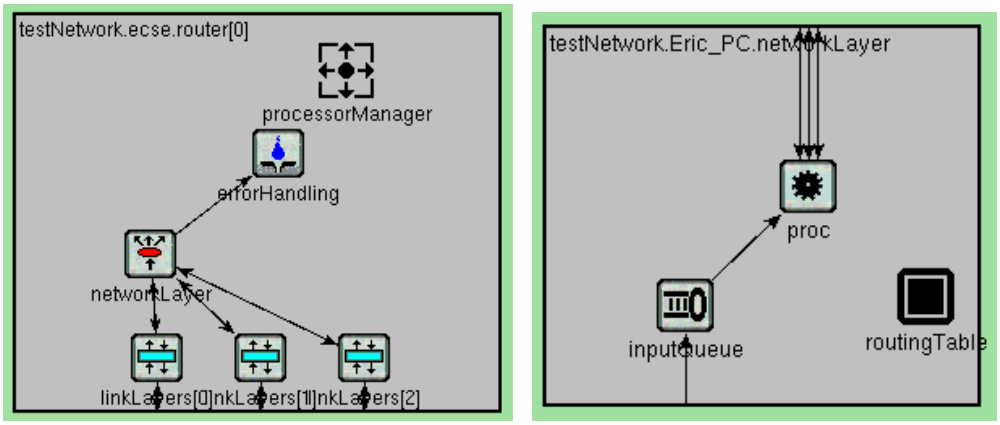
A router in an IPv6 network has many configurable parameters. We believe that the user should not have to learn the custom syntax of a configuration file in order to change a single parameter. From a user's point of view any approach that can reduce the learning curve involving a new simulation tool will be very useful. For this reason, we have chosen *Extensible Markup Language* (XML) as the format for the configuration file of the network nodes. The reasons behind our decision can be summarized as follows: XML is easy to comprehend, non-proprietary and mature technology with many tools available (parsers, viewers and validators etc.).

3.1 IPv6 Node Hierarchy

The architectural framework of the IPv6 simulation model is based on the structure of the OMNeT++ IPv4 Protocol Suite [14]. The IPv4 suite consists of modules that model the data link control, network and transport (TCP and UDP) layers. Our IPv6 simulation model framework is interoperable with the IPv4 models to support modeling dual stack routers which allow IPv4 and IPv6 packet flows simultaneously. The suite also allows various data link control layer network interfaces to be present within a single node. Therefore, it is possible to investigate the interactions between IPv4 and IPv6 protocols in a mixed protocol environment. We believe that the introduction and integration of IPv6 into the current global IPv4 infrastructure will raise performance issues that need to be investigated in large scale simulated networking scenarios.

3.1.1 Network Layer

We have adopted a different approach than the design of the IPv4 Protocol Suite [14], and separated the network interface from the network layer. This approach has allowed us to add new models of physical interfaces and to simulate routers that can have a combination of various network cards. In our simulation suite, the network layer contains only the IP processing, IP input queue and the IPv4 routing table modules (Figure 2(b)). The main functional blocks of the IP processing module are as



(a) Top level modules.

(b) Structure of the Network Layer module.

Figure 2: The simulation model of a router with three network interfaces.

follows (see Figure 3): The IP discriminator (`ipd`) module checks the IP version and forwards the packets to the correct IP stack. The IP combine (`ipc`) receives the packet from either the IPv4 or IPv6 stack and forwards the packet to the data link control layer.

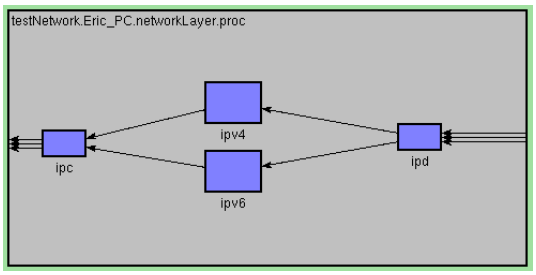


Figure 3: The main functional blocks of the IP processing module (which is part of the network layer shown in Figure 2(b)). See text for details.

3.1.2 Data Link Control Layer

The Data link control layer module shown in Figure 4 contains the input queue and an interchangeable network interface. This arrangement allows one to accommodate different physical transports without a need of recompilation of simulation models. At the time of writing, PPP and Ethernet interfaces have been implemented. The Ethernet model also includes a hub which is derived from the work of Baresi [1].

3.2 Processing IPv6 Datagrams

The core functionality of the IPv6 Simulation model is implemented in the IPv6 processing compound module (`ipv6` block in `proc` which is part of the network layer). See the Figures 2(b), 3 and 5. This module determines the destination of packets, initiates and receives ICMP notifications, and implements Neighbour Discovery mechanisms.

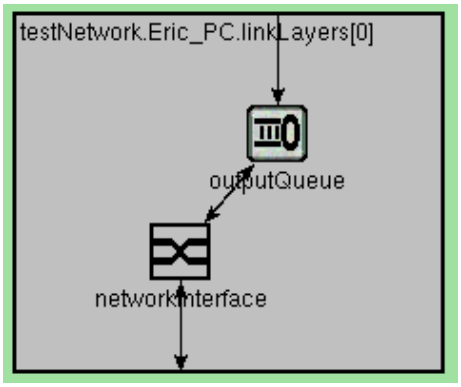


Figure 4: Structure of the simulation model for generic data link control layer modules.

Referring to Figure 5, the compound module *IPv6Processing* consists of the following submodules: *PreRouting6* (`preRouting`), *IPv6LocalDeliver* (`localDeliver`), *Routing6* (`routing`), *IPv6Multicast* (`multicast`), *AddressResolution* (`addrResln`), *ICMPv6* (`ICMP`), *IPv6Send* (`send`), *IPv6Output* (`output`), *IPv6Fragmentation* (`fragmentation`) and *RoutingTable 6* (`RoutingTable6`).

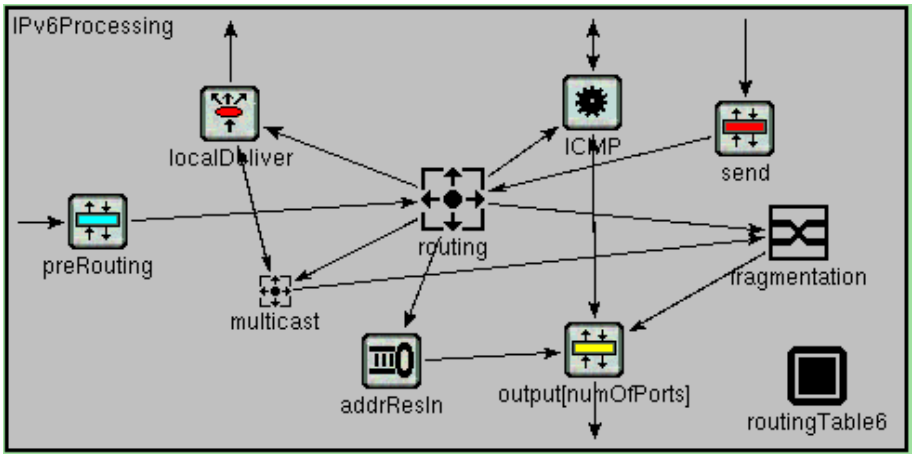


Figure 5: Internal structure of the compound module *IPv6Processing*.

Datagrams arriving a node will encounter the *PreRouting6* module first. In this module, a hook can be implemented to gather statistics or filter packets as described in [14]. Next hop determination is the responsibility of the *Routing6* module. Its options are:

- Send the datagram to an output interface via the fragmentation module when forwarding of packets is in effect i.e., it is a router,
- Send the datagram to multicast module when the packet has a multicast destination address,
- Send the datagram to localDeliver module for local delivery of the datagram.

LocalDeliver accepts datagrams destined for the local node, decapsulates the datagram and delivers its contents to upper layers. Any destination options encountered in the datagram are also processed here.

The *AddressResolution* module queries neigbhours for their data link control layer address and responds to the same requests issued by neighbouring nodes. It aims to follow the prescribed procedures defined in RFC 2461 [7] as closely as possible.

Determination of the next hop neighbour is accomplished in `Routing6` as mentioned previously with the aid of the simple module *RoutingTable6*, which contains the conceptual data structures mentioned in Section 5.2 of RFC 2461 [7]. Many other simple modules rely on *RoutingTable6* to provide access to those structures, notably *NeighbourDiscovery*, *Multicast* and *AddressResolution*.

IPv6Send encapsulates the upper layer SDUs into IPv6 datagrams and sends them to *Routing6* for further processing.

The *IPv6Fragmentation* module accepts outgoing datagrams from *Routing6* and checks to see if fragmentation is required before transferring the packets to *IPv6Output* module.

ICMP packets are managed by the *ICMPv6* compound module. The internal structure of this module is shown in Figure 6. It contains three simple modules *ICMPv6Core*, *NeighbourDiscovery* (`nd`) and *ICMPCombine* (`combine`). The *ICMPv6Core* module implements most of the RFC 2463 [2].

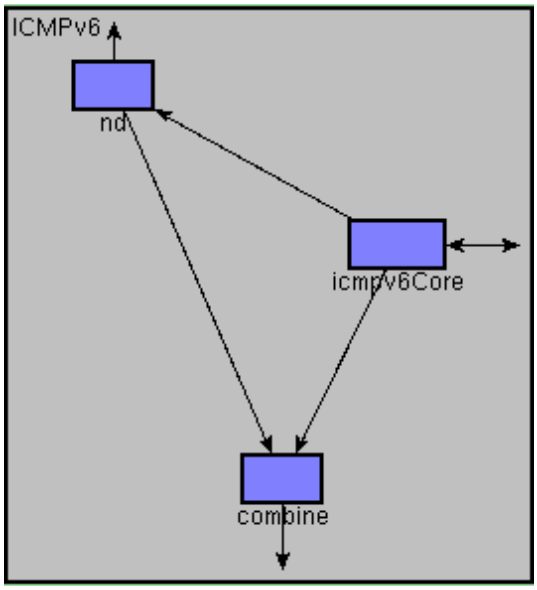


Figure 6: Components of the ICMP compound module.

The *NeighbourDiscovery* simple module initiates and responds to neighbour discovery messages according to the role of the node (host or router) in conformance with RFC 2461 [7]. *AutoConfiguration* has also been added in accordance with RFC 2462 [10].

The accuracy of the simulation is ensured due to the fine-grained level of detail in the simulation. Datagrams are passed between network entities and SDUs exchanged between the adjacent protocol layers. The IPv6 datagram currently implements most of the extension headers mentioned in [3] except the Authentication and Encapsulating Security Payload headers. Real data from simple network testbed is used to calibrate the model. Simulated processing delays are introduced where necessary to account for the differences without sacrificing performance.

3.3 Node Configuration and Parameter Specification Files

The network configuration (i.e. the connections between the network entities) is described through OMNeT++'s NED language. In addition to this, for each IPv6 node, a set of parameters can be configured by writing an XML document. (A sample XML document is shown in Figure 7). There are two ways to configure parameters of a node. In the `<global>` section, a parameter for all interfaces of all nodes on the same network is set, and in the `<local>` section, a particular parameter on a specific interface of the node is set.

4 Concluding Remarks and Future Work

Future enhancements in the pipeline include adding support for mobility; profiling the model for very large scale network simulations (i.e. 10,000+ network entities) and dynamic creation of network topologies through XML configuration file.

5 Acknowledgment

This work is supported through a Victorian Partnership for Advanced Computing (VPAC) expertise grant.

References

[1] M. Baresi. `EtherDemo` – a simple ethernet (802.3) simulation. URL reference: http://whale.hit.bme.hu/cgi-bin/contrib.pl?dir=models&txt=EtherDemo-1.0.

[2] A. Conta and S. Deering. RFC 2463 Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification, 1998. URL reference: http://www.faqs.org/rfcs/rfc2463.html.

[3] S. Deering and R. Hinden. RFC 2460 Internet Protocol, Version 6 (IPv6), 1998. URL reference: http://www.faqs.org/rfcs/rfc2460.html.

[4] D. Hasken and E. Allen. RFC 2472 IP Version 6 over PPP, 1998. URL reference: http://www.faqs.org/rfcs/rfc2472.html.

[5] R. Hinden and S. Deering. RFC 2373 IP Version 6 Addressing Architecture, 1998. URL reference: http://www.faqs.org/rfcs/rfc2373.html.

[6] U. Kaage, V. Kahmann, and F. Jondral. An OMNeT++ TCP model. In *Proceedings of the European Simulation Multiconference (ESM'2001)* [9].

[7] T. Narten, E. Nordmark, and W. Simpson. RFC 2461 Neigbhour Discovery for IP Version 6 (IPv6), 1998. URL reference: http://www.faqs.org/rfcs/rfc2461.html.

[8] OMNeT++ object-oriented discrete event simulation system. URL reference: http://www.hit.bme.hu/phd/vargaa/omnetpp.htm, 1996.

[9] The Society for Modeling and Simulation International (SCS). *Proceedings of the European Simulation Multiconference (ESM'2001)*, Prague, Czech Republic, June 2001.

[10] S. Thomson and T. Narten. RFC 2462 IPv6 Stateless Address Autoconfiguration, 1998. URL reference: http://www.faqs.org/rfcs/rfc2462.html.

[11] A. Varga. *OMNeT++ User Manual*. Department of Telecommunications, Technical University of Budapest, 1997. URL reference: ftp://ftp.hit.bme.hu/sys/anonftp/omnetpp/doc/usman.pdf.

[12] A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM'2001)* [9].

[13] A. Varga and B. Fakhamzadeh. The K-Split algorithm for the PDF approximation of multi-dimensional empirical distributions without storing observations. In *Proceedings of the* 9^{th} *European Simulation Symposium (ESS'97)*, pages 94–98, Passau, Germany, October 1997. The Society for Modeling and Simulation International (SCS).

[14] K. Wehrle, J. Reber, and V. Kahmann. A simulation suite for internet nodes with the ability to integrate arbitrary quality of service behavior. In *Proceedings of the Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS'2001)*, Phoenix, Arizona, USA, January 2001.

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <!DOCTYPE netconf SYSTEM "netconf.dtd">
3
4  <netconf>
5    <global>
6      <gAdvSendAdvertisements>on</gAdvSendAdvertisements>
7      <gMaxRtrAdvInterval>1700</gMaxRtrAdvInterval>
8      <gMinRtrAdvInterval>500</gMinRtrAdvInterval>
9      <gAdvManagedFlag>on</gAdvManagedFlag>
10     <gAdvOtherConfigFlag>on</gAdvOtherConfigFlag>
11     <gAdvLinkMTU>5644</gAdvLinkMTU>
12     <gAdvReachableTime>33567</gAdvReachableTime>
13     <gAdvRetransTimer>5346</gAdvRetransTimer>
14     <gAdvCurHopLimit>457457</gAdvCurHopLimit>
15     <gAdvDefaultLifetime>8000</gAdvDefaultLifetime>
16     <gHostLinkMTU>1400</gHostLinkMTU>
17     <gHostCurHopLimit>2</gHostCurHopLimit>
18     <gHostBaseReachableTime>232</gHostBaseReachableTime>
19     <gHostRetransTimer>234</gHostRetransTimer>
20     <gHostDupAddrDetectTransmits>1</gHostDupAddrDetectTransmits>
21   </global>
22   <local node="host1">
23     <interface>
24       <inet_addr scope="global">435:345:4:0:260:97ff:0:1/64</inet_addr>
25     </interface>
26   </local>
27   <local node="router">
28     <interface>
29       <inet_addr scope="link">fe80:0:0:0:260:97ff:0:5/64</inet_addr>
30       <AdvSendAdvertisements>off</AdvSendAdvertisements>
31       <MaxRtrAdvInterval>1231</MaxRtrAdvInterval>
32       <MinRtrAdvInterval>344</MinRtrAdvInterval>
33       <AdvManagedFlag>off</AdvManagedFlag>
34       <AdvOtherConfigFlag>off</AdvOtherConfigFlag>
35       <AdvLinkMTU>250</AdvLinkMTU>
36       <AdvReachableTime>1888</AdvReachableTime>
37       <AdvRetransTimer>222</AdvRetransTimer>
38       <AdvCurHopLimit>10</AdvCurHopLimit>
39       <AdvDefaultLifetime>6710</AdvDefaultLifetime>
40       <AdvPrefixList>
41         <AdvPrefix>3018:FFFF:0:0:0:0:0:0/48</AdvPrefix>
42       </AdvPrefixList>
43       <HostLinkMTU>60</HostLinkMTU>
44       <HostCurHopLimit>5</HostCurHopLimit>
45       <HostBaseReachableTime>500</HostBaseReachableTime>
46       <HostRetransTimer>400</HostRetransTimer>
47     </interface>
48   </local>
49 </netconf>
```

Figure 7: An example of a configuration file used for specifying the values of several parameters of the nodes in an IPv6 network.