Practical Considerations for Extending Network Layer Models with OPNET Modeler

Abstract

The Internet is an evolving force that contributes to rapid economic expansion worldwide. However, each newly emerging Internet technology requires rigorous evaluation and testing; a process that often includes simulation and modeling. OPNET Modeler is among the foremost software products for the simulation and modeling of communication protocols and Internet technologies. However, OPNET's considerable amounts of source code and supporting application programming interfaces (API) can be quite overwhelming, even for experienced developers. This paper attempts to demystify the process of modeling in OPNET and enumerate the key steps for creating new simulation models using the OPNET Modeler software package. In this paper, the authors share experiences from their foray into developing a new IP layer mechanism for QoS support. Additionally, this paper details a methodology for expanding the OPNET Modeler network layer implementation.

Keywords: modeling, simulation, OPNET, protocols, network layer, IP, QoS.

1. Introduction

According to the Organization for Economic Co-operation and Development, information and communication technologies are growing rapidly and have a significant impact on the global economy [1]. The Internet is among the key contributing factors that compel this rapid economic expansion. Internet technologies that were once exotic, such as web-casting, Internet TV, and voice over IP, have become parts of everyday life. However, before a new technology can be released to the public, it must pass rigorous testing and evaluation; a process that often involves simulation and modeling.

The OPNET Modeler software package [2] is among the most popular and most comprehensive tools available on the market for modeling new communication technologies and protocols. OPNET Modeler includes a vast model library of communications devices, communication mediums, and cutting-edge protocols. It also allows users to extend models and create new ones using C/C++.

However, developing or expanding simulation models is a quite complex and challenging task. When working with OPNET Modeler, developers are faced with considerable amounts of source code and supporting application programming interfaces (API). This can be quite overwhelming, even for experienced software developers. Often the hardest question to answer is, "Where do I start?" In this paper, we attempt to demystify the process of modeling in OPNET as well as enumerate the key steps for creating new simulation models using the OPNET Modeler software package. In particular, we concentrate on network layer technologies and protocols, including mechanisms for quality of service (QoS) support [3] over Internet Protocol (IP) [4].

In this paper, we share experiences from our foray into developing a new IP layer mechanism for QoS support, and present a methodology for expanding the OPNET Modeler network layer implementation. The simulation models described in this paper have been implemented with version 11.5 of OPNET Modeler. Upon this writing, the latest release of OPNET Modeler is version 12.0, which has been available since late 2006. However, version 12.0 includes no significant changes to the network layer implementation. Thus, the processes and methodologies described in this paper are applicable to both version 11.5 and 12.0 of OPNET Modeler.

The rest of the paper is organized as follows. Section 2 provides brief overview of the OPNET Modeler architecture followed by Section 3 that describes methodology for developing and integrating new models in OPNET. Section 4 presents an example of using described methodology to implement. Summary and conclusions appear in Section 6.

2. Overview of OPNET Architecture

The OPNET Modeler architecture consists of three modeling domains: the process, the node, and the network. Within the process modeling domain the developer implements the behavior of various

processes, such as e-mail clients, TCP managers, or IP interfaces. In OPNET, the modular implementations of these processes are referred to as process models. The complete specification of an OPNET process model consists of a finite state machine, action statements expressed in C/C++, and configurable parameters.

Within the node modeling domain the developer implements the behavior of various network devices, such as clients, servers, switches, or routers. Node models are usually defined via one or more functional elements called modules and by the data flow between them. The behavior of individual modules is specified either via a set of built-in parameters or through one or more process models.

Within the network domain the developer implements complete network models including individual nodes and interconnecting communication links. A network model specification also includes the configuration of such simulation model characteristics as individual applications, user profiles, and network protocols. The configuration of individual nodes and communication mediums, their connectivity and geographical locations, serves to further define a network model. The attribute values specified in the network modeling domain propagate all the way down to the process models. The attribute values specify either local characteristics, applicable to individual devices, or global characteristics, applicable to multiple devices in the network.

The OPNET modeling architecture is structured in a layered fashion with the process domain being the lowest layer, followed by the node domain, and finally the network domain layer at the top. The process models are used directly to build node models, which in turn are combined to build various network models.

3. Methodology for adding a new process models

Generally, in OPNET Modeler, creating new process model consists of three major steps: specifying simulation-wide attributes (if needed), developing a process model of desired technology or protocol, and finally integrating the process model within OPNET. The rest of this section examines these steps in more detail.

3.1. Specifying simulation-wide attributes

It is appropriate to define an attribute that contains the same value in all applicable nodes in the network as simulation wide. For example, such attributes as application definitions, user profiles, and active queue management disciplines are easier to configure through a single configuration node than by setting up the same definition in every node of the network model. To define simulation-wide attributes the developer should modify one of the corresponding process models. For example, the developer should modify the ***application_config*** process model when creating new application definitions, while the ***profile_config*** and ***qos_attribute_definer*** process models contain simulation-wide attributes for configuring user profiles and various QoS mechanisms, respectively.

The process models for specifying simulation-wide attributes are simple and easy to expand. They usually consist of one of more initialization states that parse the attribute values and store them into simulation-wide database using facilities of the OPNET Model Support (OMS) data definition package, ***oms_data_def***. During simulation, these process models are instantiated first, which means that the user-defined process model is invoked when the simulation-wide attributes are already stored in the database and can be retrieved using the corresponding procedure calls. The ***oms_data_def*** package contains two procedures: *oms_data_def_entry_insert* to write the data entries into and *oms_data_def_entry_access* to retrieve the data entries from the simulation-wide database.

3.2. Developing new process models

The steps required for creating new process models depend solely on the nature of the technology or communication protocol being implemented. However, there are several steps that are common for the majority of process models. These include collecting and registering statistics, setting-up model attributes, and providing inter-process communication.

Process model statistics provide feedback to the user about the behavior and performance of the model. For example, the process model may collect such statistics information as the number of bytes sent, the number of packets dropped, and others. There are two types of statistics available in OPNET: local, which are computed for each process model separately; and global, which are computed as a sum or average for all instances of the same process model within the simulation. Each statistic, local or global, has to be declared via the Process Editor GUI and then registered using the *op_stat_reg* procedure during initialization of the process model. Once a statistic has been registered, it can be updated as needed using the *op_stat_write* procedure.

Model attributes communicate information from the network domain to the process model. For example, the process model of the Transport Control Protocol (TCP) [5] contains attributes such as the receiver window size and the maximum segment size. These attributes have their values set within the network domain but determine TCP's behavior at the process level. There are two attribute types: local, i.e. each instance of a process model has its own set of local attributes, and global, i.e. attributes that are common to all process model instances within a simulation. As with the statistics, the attributes have to be declared via the Process Editor GUI before they can be used. After that, the user-specified values can be read, usually during the process model initialization, using function *op_ima_obj_attr_get*. One of the best ways to learn how to set-up and use process model statistics and attributes is by examining other OPNET process models such as ***application_config, ip_dispatch***, ***tcp_manager_v3, rip_udp_v3***, and others.

Once the process model statistics have been declared and the model attribute values have been obtained, the developer can start implementing the behavior of the new technology or protocol. Often to achieve the desired effect the new technology requires exchange of information among the nodes in the network. For example, to support quality of service requirements the Integrated Services [6] rely on the Resource Reservation Protocol (RSVP) [7]. RSVP forwards control messages about the flow requirements and resource availability between the routers. Modeling such message exchange in OPNET is quite challenging. There are several approaches to implementing communication between nodes in OPNET. In this paper we concentrate on two approaches to: explicit exchange of control packets and use of a simulation-wide database.

To implement inter-node communication using the explicit exchange of control packets, the process model has to use OPNET facilities for generating and processing packets. The **PK** (Packet) package, described in Chapter 11 of the Discrete Event Simulation API Reference Manual [8], provides a collection of the procedures for manipulating packets in OPNET. Modeling inter-node communication using explicit packet exchange enables precise representation of the protocol behavior, and will accurately reflect such performance characteristics as delay of the packets as they travel through the network. However, implementing explicit packet exchange is usually a complex, challenging, code-intensive, and often error-prone task.

The use of simulation-wide database is more appropriate in the situations when the exchange of protocol control information can be separated from the rest of the design and control packet latency can be neglected. Even though this is not intended use for the OMS facility, the ***oms_data_def*** package can be used to exchange information between nodes as well. When needed, the control information can be stored in a model-specific data structure and then written using the *oms_data_def_entry_insert* procedure into the simulation-wide database. Similarly, the control information can be retrieved from the simulation-wide database using the *oms_data_def_entry_access* procedure and then stored in the model-specific data structure. This approach significantly reduces implementation complexity of the process model but may result in inaccurate simulation results because the control packet propagation through the network and associated delays are not simulated.

3.3. Integrating the new process model within OPNET

Integrating a new process model within OPNET is a challenging and often confusing task because it requires intimate knowledge of the standard, built-in OPNET process models, which are quite complex. If the process model implements a new application then the developer needs to understand

the inner-workings of the generic client-server manager (***gna_clsvr_mgr***) process model; to implement or modify a routing protocol the developer requires an understanding of the corresponding routing protocol process model; and so on. This knowledge is needed to determine how to invoke the new process model from within the OPNET's code. Describing the details of all existing standard OPNET process models is beyond the scope of this work. Instead the paper concentrates on the implementation of the IP protocol and related technologies which are the heart of the Internet.

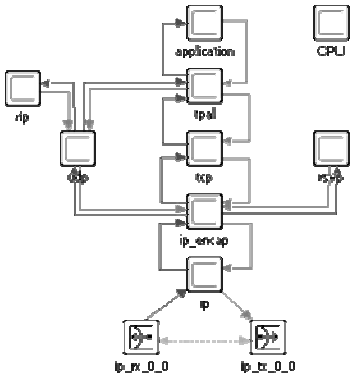

Figure 1. Node model of a point-to-point workstation

Figure 1 illustrates an OPNET node model of a point-to-point workstation, where grey boxes represent modules that implement various protocols, and the arrows designate the inter-module communication. As Figure 1 shows, the network layer is simulated via two modules: ***ip_encap*** and ***ip***. The ***ip_encap*** module uses the ***ip_encap_v4*** process model to implement encapsulation, e.g. adding the IP header to the packets that travel from the upper layers down into the network, and decapsulation, i.e. stripping the IP header off the packets that arrive from the network and travel to the upper layers. The ***ip*** module relies on the ***ip_dispatch*** process model to implement such basic IP activities as forwarding, fragmentation, and reassembly. In addition, ***ip_dispatch*** invokes the ***ip_output_iface*** process model to implement packet processing at individual outgoing interfaces of the node. The majority of new IP technologies for QoS support are based on effective management of outgoing interfaces and network resources associated with them, such as bandwidth and queue occupancy. In OPNET, such technologies are implemented via the ***ip_output_iface*** process model which we examine next.

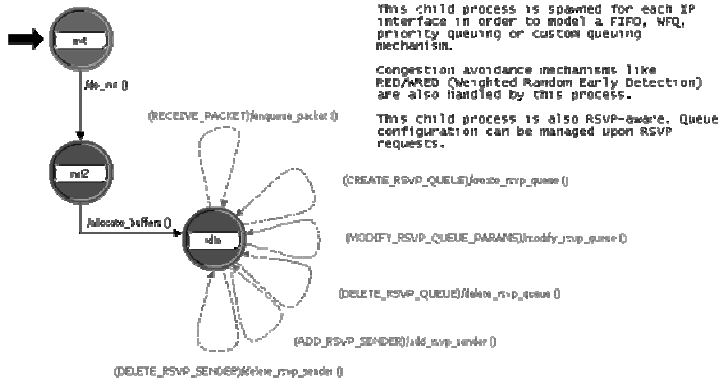

Figure 2. Process model ***ip_output_iface***

As Figure 2 illustrates the ***ip_output_iface*** process model consists of three states. States "init" and "init2" initialize all of the process model's variables and data structures in the *do_init()* and *allocate_buffers()* procedures. State "idle", on the other hand, deals with packet arrivals and the RSVP protocol. We ignore the RSVP implementation because it is beyond the scope of the paper. We primarily concentrate on initialization and packet processing which are key procedures for adding new network layer process models.

First, the developer needs to call a procedure to spawn a child process for the new process model. This can be accomplished by calling the *op_pro_create()* OPNET kernel procedure, which returns the process handle of, or the reference to, the created child process. The new process model should be spawned during the ***ip_output_iface*** process model initialization. The *do_init()* procedure deals with the initialization of various process model variables and data structures. While the *allocate_buffers()* procedure deals with the initialization of the output interface buffers used to model various active queue management and scheduling disciplines. That is why, from the software engineering point of view, the *do_init()* procedure is more appropriate than the *allocate_buffers()* procedure for spawning new processes.

Once the new process is created, it can be invoked, as necessary, using the *op_pro_invoke()* OPNET kernel procedure. Usually a new process invocation is associated with a packet arrival or departure. Generally, there are two main events that occur in the ***ip_output_iface*** process model and that may trigger a new process invocation: (1) the packet that has not been processed yet arrives on the outgoing interface and (2) the packet is processed and is scheduled for departure from the outgoing interface. From the networking point of view it is logical to invoke a new process when the packet arrives on the outgoing interface and before the packet is placed in the queue. In such cases the new process has a chance to examine every packet that arrives on the interface. If the new process is invoked upon the packet departure, then not all of the packets may be accessible because some packets may be discarded upon queue overflow. To invoke the new process before the packet is placed in the queue the developer should call the *op_pro_invoke()* kernel procedure from the *enqueue_packet()* procedure. The *enqueue_packet()* procedure is executed each time the packet arrives on the outgoing interface.

On the hand, one can envision situations where the new process has to be invoked after the packet has been processed. For example, the new process deals only with those packets that were not discarded and are ready to be placed on the physical wire. In this case, the developer should modify procedure *extract_and_send()* which is called after queue management (QM) completed packet processing.

Once the new process is not longer needed, the developer can terminate it using the OPNET kernel procedure *op_pro_destroy()*. This kernel procedure can be called either from the process model's termination block, or from the *enqueue_packet()* or *extract_and_sent()* procedure upon arrival or departure of certain packet. It should be noted that all process model procedures such as *do_init(), allocate_buffers(), enqueue_packet(), extract_and_sent(),* and others can be examined and modified via the process model's function block of a process model.

Editing or adding new queue management and scheduling disciplines requires a modification of the QM package procedures stored in the *oms_qm_ex.c* file. In particular, the developer would need to study and update such procedures as *Oms_Qm_Packet_Process(), Oms_Qm_Incoming_Packet_Handler(), Oms_Qm_Packet_Enqueue(),* and others. However, this discussion is beyond the scope of the work and is not included in the paper.

4. Example of the network layer technology implementation in OPNET

This section discusses implementation of the Bandwidth Distribution Scheme (BDS) in OPNET to serve as an example of how develop network layer process models and integrate them in OPNET Modeler.

4.1. Brief overview of the Bandwidth Distribution Scheme

The Bandwidth Distribution Scheme (BDS) is a mechanism for supporting quality of service in the Internet. In short, BDS operates as follows. Each new flow that enters the network specifies the minimum requested rate which is the minimum amount of bandwidth that the flow needs to operate properly, and the maximum requested rate which is the maximum amount of bandwidth that the flow can utilize. These values are called the Requested Bandwidth Range (RBR) and all admitted flows are guaranteed to receive the amount of bandwidth within its RBR. If there is a sufficient amount of bandwidth to support the new flow's request then all the other flows that share resources with the

newly admitted flow are throttled to accommodate the new flow arrival. The core nodes in the network maintain the sum of the flow RBR values for each of their outgoing links, which is called an aggregate RBR. Aggregate RBR values are distributed among edge nodes for the purpose of computing the fair share transmission rates of each flow. The edge nodes monitor the flow arrival rate and throttle the flows, if needed, by discarding those packets that arrive at the rate above their computed fair share. BDS relies on a fairly complex message exchange protocol for maintaining the aggregate RBR values in the network core and distributing them among the edge routers. BDS operates at the network layer and is implemented as an extension of the outgoing interface processing in the IP layer. Refer to [9, 10] for additional details on BDS operation.

4.2. Summary of BDS Implementation in OPNET

First step towards implementing BDS was to modify the ***qos_attribute_definer*** process model for parsing the BDS attribute values specified at the network domain level. Parsed data is stored in the simulation-wide database using the ***oms_data_def*** package.

To support BDS we created two data structures: *flow* table and *interface* table. Each entry in the *flow* table contains such information as source and destination IP addresses (for flow identification), RBR, computed fair share, complete path to the destination, and current transmission rate. The *interface* table contains such information as available link capacity for BDS traffic, and the aggregate RBR values for the flows that travel via the interface. In real a network, such data structures are not available globally and are local to individual nodes. However, since the goal of our study was to examine the feasibility of BDS independently of the control packet exchange protocol, the *flow* and *interface* table information was made available to all nodes in the network using simulation-wide database of the ***oms_data_def*** package.

We implemented the bandwidth distribution scheme as a separate process model invoked from the ***ip_output_iface*** process model upon the packet arrival. Figure 3 illustrates the BDS process model. The "BDS_INIT" state initializes various local data structures including the *flow* and *interface* tables. Once the initialization is complete the process model transitions into the "BDS" state, were it waits for a packet arrival.

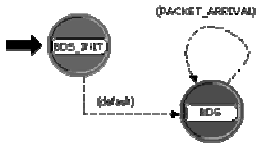

Figure 3. BDS process model.

When a packet arrives, BDS retrieves packet information provided by the ***ip_output_iface*** process, identifies the flow the packet belongs to based on source and destination IP addresses, and retrieves flow information from the *flow* table. Next, BDS updates the corresponding entries in the *flow* and *interface* tables using the *oms_data_def_access()* and *oms_data_def_insert()* procedures.

Finally, we integrated the BDS process model within the ***ip_output_iface*** process model. We spawned the BDS process by calling the *op_pro_create()* kernel procedure from the *do_init()* procedure. We invoke the BDS process upon each packet arrival from the *enqueue_packet()* procedure by calling *op_pro_invoke()* kernel procedure. Upon completion the packet processing, BDS returns a Boolean variable set to TRUE if the packet arrived at the rate above its flow's fair share and FALSE otherwise. We also modified the *enqueue_packet()* procedure to examine the value returned by the BDS process before placing the packet in the queue. If the value is TRUE then the packet is discarded using the kernel procedure *op_pk_destroy()*, otherwise the packet is placed in the queue as usual.

The BDS implementation was tested using several different scenario settings. Preliminary results indicate that the current BDS implementation distributed available bandwidth resources within 2% of expected fair share values. However, a complete report of the study is beyond the scope of the paper.

We are currently completing the result compilation, expanding the BDS implementation, and planning to present a comprehensive report of the study in future publications.

5. Conclusions

This paper examined a practical methodology for expanding network layer technologies using the OPNET Modeler software. We plan to extend this study and provide a more detailed account of the BDS implementation in our future work. We also plan to examine the OPNET implementation of the active queue management and scheduling mechanism in greater detail and implement the control packet exchange protocol during the next phase of our study.

References:

[1] Organization for Economic Co-operation and Development (OECD), Information and Communication Technologies, http://www.oecd.org/topic/0,2686,en_2649_37441_1_1_1_1_37441,00.html, last visited on 07/01/07.

[2] OPNET Inc., http://www.opnet.com/, last visited on 07/01/07.

[3] Quality of Service Networking, http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm, last visited on 07/01/07.

[4] Postel, J., "Internet Protocol", RFC 791, DARPA, September 1981.

[5] W. Feng, D. Kandlur, D. Saha, K. Shin, Understanding and improving TCP performance over networks with minimum rate guarantees, IEEE/ACM Transactions on Networking 7 (2) (1999) 173–187.

[6] R. Braden, D. Clark, S. Shenker, “Integrated Services in the Internet Architecture: an Overview”, June 1994, IETF RFC 1633.

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, S. Jamin, Resource reservation protocol (RSVP)—version 1 functional specification, September 1997, IETF RFC 2205.

[8] OPNET Modeler version 11.5. Discrete Event Simulation API Reference Manual

[9] V. Hnatyshin and A. S. Sethi, “Bandwidth Distribution Scheme for Dynamic, Scalable, and Fair Allocation of Bandwidth,” International Journal of Network Management (Wiley), Volume 16, Issue 5, September/October 2006, pp 317- 336.

[10] V. Hnatyshin and A. S. Sethi, “Scalable Architecture for Providing Per-flow Bandwidth Guarantees," Proceedings of the IASTED conference on Communications, Internet and Information Technology (CIIT 2004), St. Thomas, Virgin Isles, November 2004