OPEN ACCESS

International
Journal
Of Modern Engineering Research (IJMER)

Design of an Ns-3 Generic Application Architecture Applying Design Patterns

Jong Min Lee*
**Dept. of Computer Software Engineering, Dong-Eui University, Busan, Republic of Korea*

ABSTRACT:- *Ns-3 is an object-oriented event-driven network simulator which has been developed using the pure C++ language. There are some existing applications modeling FTP and CBR traffics as traffic generating applications for networking research. However it is not efficient to develop a new application using the inheritance when it needs a new traffic pattern. In this paper, we propose a new application architecture by performing the commonality/variability analysis for the existing application classes and then applying design patterns to make a flexible and extensible architecture. It is expected that we can develop a new application for a new traffic pattern without any duplicated code by enhancing the problem due to the inheritance.*

Keywords: *Ns-3, traffic generation, commonality analysis, strategy pattern, template method pattern*

I. INTRODUCTION

Ns-3 is the discrete event network simulator developed for the research and education replacing the existing ns-2 network simulator [1]. The ns-2 network simulator had been developed using the object-oriented programming language C++ and OTcl [2]. Since 2006 ns-3 has been developed as an open source project. It can be used as both a research-oriented simulator and a real target device to generate packets in association with the Internet. Ns-3 has been implemented purely using C++ and it supports an interface for the Python script language to use it with ease.

In this paper, we focus on the architecture of classes related to the network traffic generation such as FTP and CBR (constant bit rate). The existing architecture uses the inheritance of ns3:: Application to add a new application and thus it overrides operations for a socket connection and an application-dependent traffic generation, which is not efficient because it produces many duplicated codes without reusing existing codes. In this paper, we propose efficient traffic-generation architecture by improving the architecture of the existing application-relevant classes.

In Section 2, the basic architecture of ns-3 and important classes for the network research are described. In Section 3 and Section 4, the proposed application architecture and the evaluation result are described respectively. Finally, the concluding remarks are given in Section 5.

II. THE EXISTING NS-3 ARCHITECTURE

In this section, we describe the software architecture of ns-3 by reverse-engineering the source code using Enterprise Architect of Sparx Systems [3].

2.1 ns-3 Components

Components used for networking in ns-3 are classes such as ns3::Node, ns3::Application, ns3::Channel, ns3::NetDevice and several Helper classes for helping the configuration of these classes and the connection. Fig. 1 shows the class diagram of basic components of ns-3. Several ns3::Application objects can be connected to one ns3::Node object to generate and receive traffic packets, which can be transmitted through ns3::NetDevice objects and ns3::Channel objects.

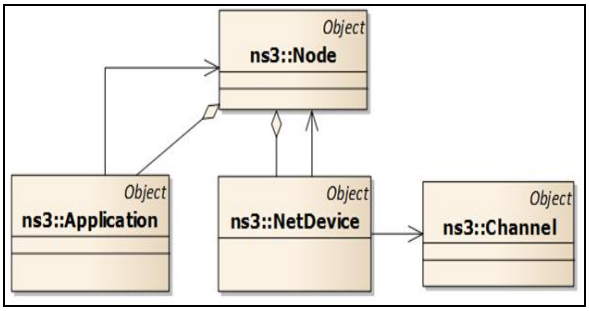

Figure 1. Basic components of ns-3

The object diagram is shown in Fig. 2 to give information about how traffic-relevant classes such as ns3::Application, ns3::Node, ns3::NetDevice and ns3::Channel are connected. Instances of ns3::Application, ns3::Node and ns3::NetDevice comprise both a client side and a server side. Both sides are connected through one ns3::Channel instance like the real network environment.

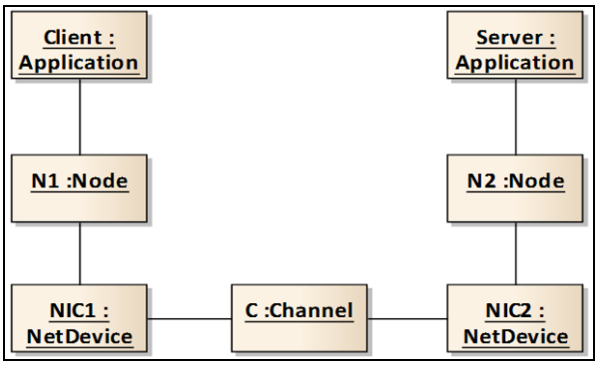

Figure 2. The relationship among traffic-related classes

Fig. 3 shows the class diagram of the FTP traffic-generation application. Several instances of ns3::Application can be attached to ns3::Node. ns3::Application has attributes to start and stop the application, m_startTime and m_stop Time. Two virtual member functions, StartApplication() and StopApplication(), are overridden by the same member functions in ns3::BulkSendApplication to define how and when to generate traffic packets. Message transfer is performed using ns3::Socket, which is a low-level Socket API based loosely on the BSD Socket API.

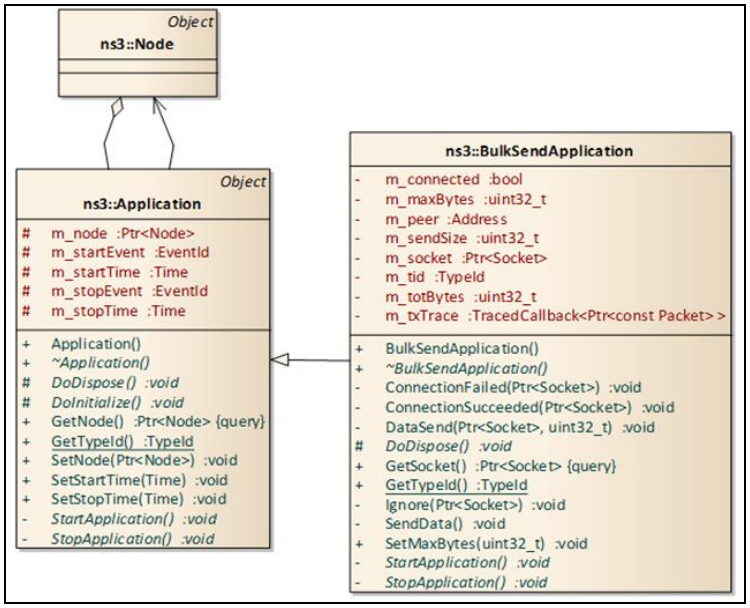

Figure 3. The FTP traffic-generating application

The ns3::Application class is used for generating traffic packets for the network simulation research. FTP and CBR traffics are basic traffic-generating applications for the network research [4]. Ns3: BulkSendApplication and ns3::PacketSink classes are used for the bursty FTP traffic and ns3::UdpClient and ns3::UdpServer classes are used for the constant bit rate (CBR) traffic, which are shown in Fig. 4.

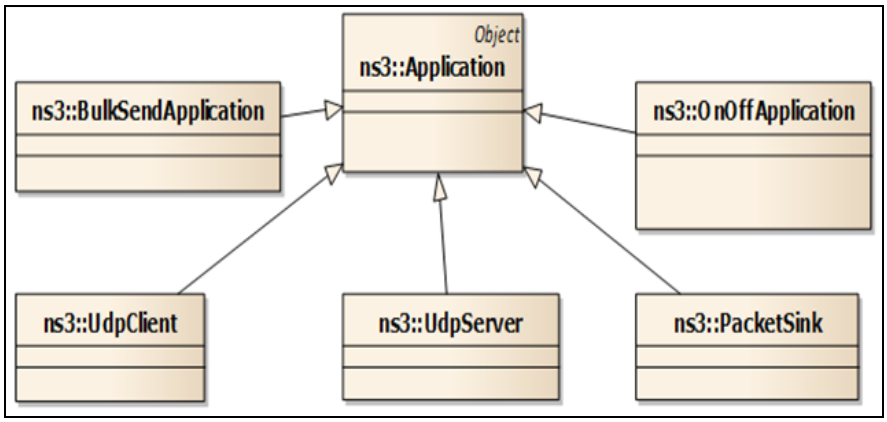

Figure 4. Ns-3 application classes

2.2 FTP Traffic

One of the most widely used traffic pattern of the network simulation is FTP (file transfer protocol or program). To generate FTP traffic packets, ns3::BulkSendApplication and ns3::PacketSink are used as a source and a destination respectively as shown in Fig. 5. To model a large-scale file transfer, attributes of ns3::BulkSendApplication are defined as in Table 1. ns3::PacketSink is used just for receiving packets through a network, which does not perform any special functionality. It is possible to get information of a designated packet using a tracing system of ns-3.

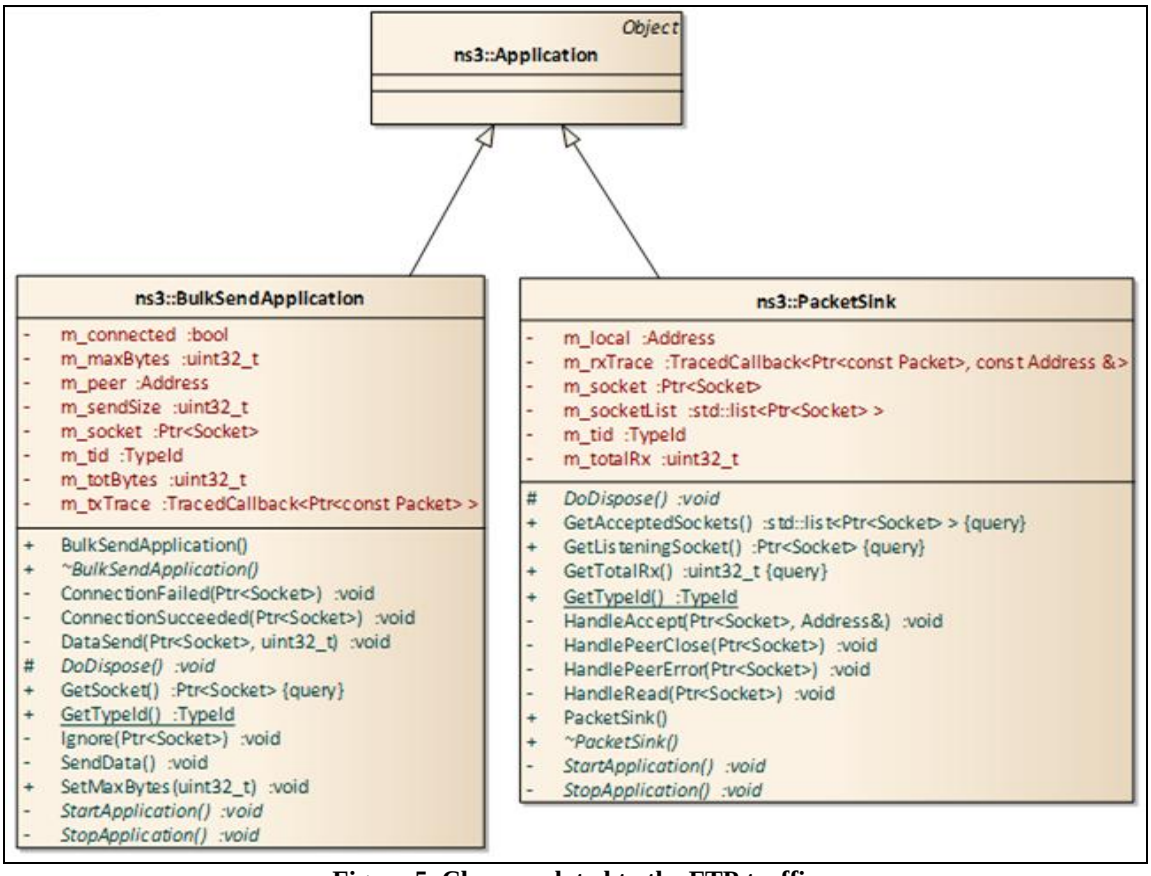

Figure 5. Classes related to the FTP traffic

Table 1. Attributes used for the FTP traffic

Attribute	Description
SendSize	the amount of data to send each time (default: 512)
MaxBytes	the total number of bytes to send. The value zero indicates no limit. (default: 0)
Remote	the destination address including a port number
Protocol	the protocol used (ns3::TcpSocketFactory (default) or ns3::UdpSockerFactory)

2.3 CBR Traffic

ns3::UdpClient and ns3::UdpServer are used to generate packets for the CBR traffic, which is shown in Fig. 6. ns3::UdpClient has the following attributes in Table 2. ns3::UdpServer has attributes Port and PacketWidowSize. Port is the receiver port number (default: 100) and PacketWindowSize is the size of a window used for calculation of packet loss (multiple of 4, possible values: 8 ~256, default: 32).

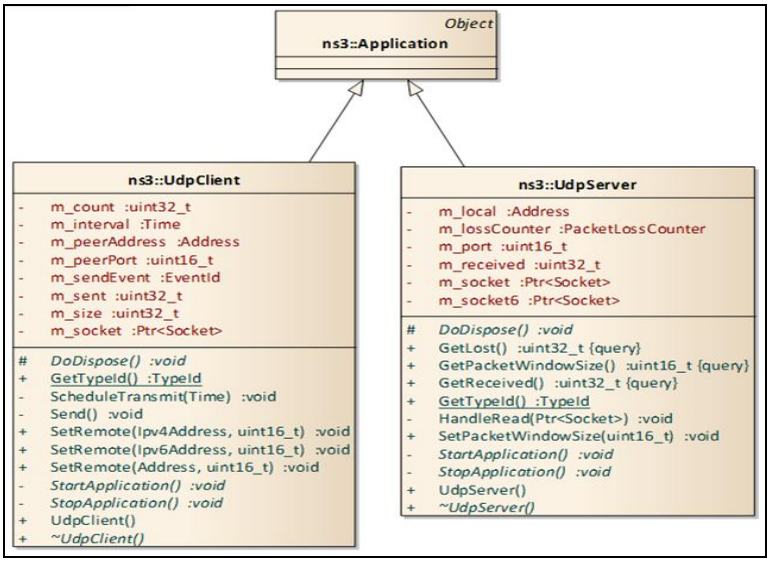

Figure 6. Classes related to the CBR traffic

Table 2. Attributes used for the CBR traffic

Attribute	Description
MaxPacket	the maximum number of packets to send (default: 100)
Interval	the time period of two consecutive packets (default: 1 second)
RemoteAddress	the destination address
RemotePort	the destination port number (default: 100)
PacketSize	the number of bytes per packet (possible values: 12~1500, default: 1024)

2.4 OnOff Traffic

ns3::OnOffApplication in Fig. 7 implements the OnOff traffic pattern which can be used in modeling the VoIP voice traffic [5]. The voice activity detection (VAD) can be used to model the speaker's speech state more precisely [6]. Like ns3::BulkSendApplication, ns3::PacketSink is used as a receiving application of a destination node. ns3::OnOffApplication has the following attributes in Table 3.

Table 3. Attributes used for the OnOff traffic

Attribute	Description
DataRate	The data rate in on state (default: 500,000bps)
PacketSize	The size of packets sent in on state (default: 500)
Remote	The address of the destination
OnTime	A RandomVariableStream used to pick the duration of the 'On' state (default: 1.0)
OffTime	A RandomVariableStream used to pick the duration of the 'Off' state (default: 1.0)
MaxBytes	The total number of bytes to send (default: 0 – no limit)
Protocol	the type of protocol to use (default: ns3::UdpSocketFactory)

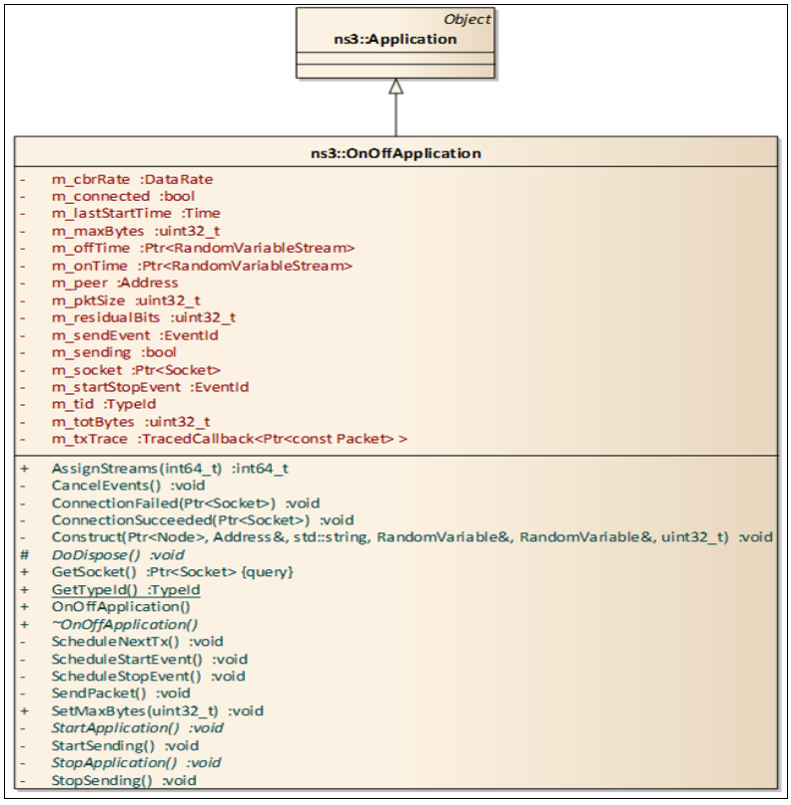

Figure 7. A class related to the OnOff traffic

III. THE PROPOSED ARCHITECTURE

In this section, we propose the architecture of a generic application by performing the commonality/variability analysis for the classes used for generating traffic packets.

3.1 Commonality/Variability Analysis

In order to find the relationship among the classes used for generating traffic packets, the commonality/variability analysis [7] in the object-oriented analysis and design is used. By performing the analysis, we propose the architecture of an application class which can generate a new traffic pattern with ease including the existing traffic patterns.

Table 4 show the result of the commonality analysis for attributes of source classes: ns3::BulkSendApplication (B), ns3::UdpClient (U) and ns3::OnOffApplication (O). The packet size and the total number of bytes to send to a destination node are common attributes and they can be defined as attributes in a proposed generic application class.

Table 4. The commonality analysis of attributes of source classes

Purpose of Use	Attributes
the amount of data to send each time	SendSize (B)
	PacketSize (U, O)
the total number of bytes to send	MaxBytes (B, O)
	MaxPackets (U)

Table 5 show the result of the variability analysis for attributes of source classes. Attributes of each class are very different from the others because of characteristics of each traffic pattern. If these attributes are implemented using the inheritance to define a new child class, the architecture cannot be flexible due to the duplicated code. To solve this problem, we use the Map collection maintaining the information in the form of <K,V>, where K and V represent a key and a value respectively. In C++, std::map<K,V> is defined to implement the Map object and java.util.Map<K,V> interface is defined in JAVA.

Table 5. The variability analysis for attributes of source classes

Class name	Attributes
ns3::BulkSendApplication	Remote
	Protocol
ns3::UdpClient	Interval
	RemoteAddr
	RemortPort
ns3::OnOffApplication	DataRate
	Remote
	OnTime
	OffTime

Table 6 shows the result of the commonality analysis of operations of source classes. StartApplication() and StopApplication() operations are overridden by inheriting ns3::Application. The name of operations related to packet generation are slightly different but its functionality is so similar that they can be merged into the same operation. There are operations related to the connection management in both ns3::BulkSendApplication and ns3::OnOffApplication, but they don't seem to have any special functionality according to the ns-3 source code.

Table 6. The commonality analysis of operations of source classes

Purpose of Use	Operations
start of an application	StartApplication (B, U, O)
stop of an application	StopApplication (B, U, O)
packet generation	SendData (B)
	Send (U)
	SendPacket (O)
connection management	ConnectionFailed (B, O)
	ConnectionSucceeded (B, O)

Table 7 show the result of the variability analysis of source classes. ns3::OnOffApplication has a lot of different kinds of operations compared to the others because behaviors of the packet transfer period and the packet non-transfer period are different. But these operations of ns3::OnOffApplication are used for scheduling the Send Packet operation in Table 6.

Table 7. The variability analysis of operations of source classes

Class name	Operations
ns3::BulkSendApplication	GetSocket
ns3::UdpClient	SetRemote
ns3::OnOffApplication	GetSocket
	StartSending
	StopSending
	ScheduleStartEvent
	ScheduleStopEvent
	ScheduleNextTx

ns3::PacketSink and ns3::UdpServer classes are used as a receiving application of a destination node. ns3::PacketSink uses basically the UDP socket but it can also use the TCP socket so that ns3::PacketSink can be used for all traffic-generating applications. ns3::UdpServer receives only packets generated by ns3::UdpClient. ns3::UdpServer has PacketLossCounter and PacketWindowSize attributes for calculating the packet loss due to the transmission error of the UDP. These receiving applications just receive packets from a source and do not seem to have much functionality to optimize, and thus we focus on how to design those packet-generating applications such as ns3::BulkSendApplication, ns3::UdpClient and ns3::OnOffApplication in this paper.

3.2 Proposed Architecture

Fig. 8 shows the class diagram of the extensible architecture proposed in this paper in order to include the existing packet-generating classes such as ns3::BulkSendApplication, ns3::UdpClient and ns3::OnOffApplication and a new application with a different traffic pattern. Basically the class 'GenericApplication' performs the same

functionalities by inheriting ns3::Application, but it can add and modify its functionalities to create a socket and generate the network traffic much easily because of its flexible architecture

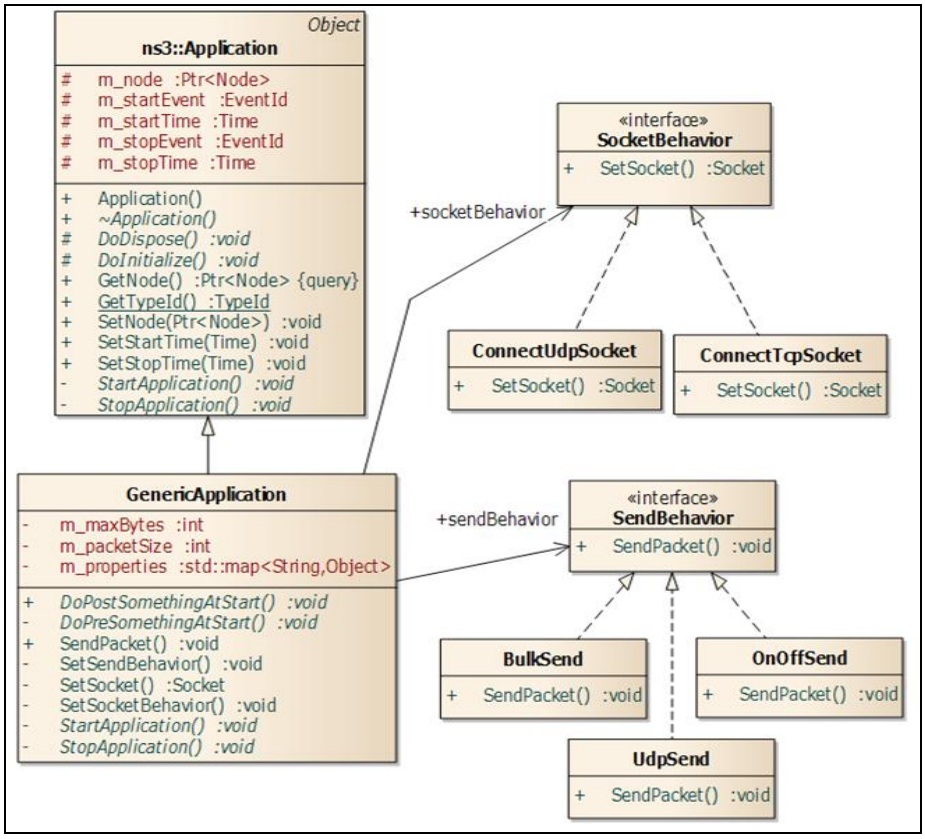

Figure 8. The class diagram of the proposed architecture applying the strategy pattern

Two operations, SetSocket and SendPacket are designed by applying the strategy pattern [8,9], by which two operations can select and execute a proper algorithm dynamically in run-time according to the given traffic pattern. The SetSocket operation can be different according to the used protocol and thus it is necessary to change the behavior in run-time. To implement like this, the interface, Socket Behavior, is defined and two classes, ConnectUdpSocket and ConnectTcpSocket, are defined by implementing Socket Behavior. As we can see by the class name, ConnectUdpSocket and ConnectTcpSocket make a connection using the UDP and TCP protocol respectively. TCP and UDP are the most widely used network protocol, which is sufficient for the network connection but it is easy to add a new network protocol like SCTP [10] by implementing SocketBehavior.

The interface, Send Behavior, is also defined to implement the behavior to send a packet according to the given traffic pattern by applying the strategy pattern, which also make the operation, Send Packet, flexible. Three classes, BulkSend, UdpSend and OnOffSend, implements this interface and perform the same functionality of ns3::BulkSendApplication, ns3::UdpClient and ns3::OnOffApplication respectively. If a new network traffic pattern is required, we can add a new class by implementing Send Behavior.

As we can see in Table 5, there are many different attributes for different source classes. To manage these different attribute, we can easily find a solution by make each corresponding subclasses to different traffic patterns. It is obvious that the inheritance is the most powerful mechanism of the object-oriented programming. In this case, however, it is not efficient in terms of the code reusability because we have to make a new child instance when we need a new network traffic pattern. Attributes should not be the main cause of using the inheritance. The map [11,12] should be used in this case to handle different attributes of different classes efficiently [7], which makes it possible to handle different kinds of attributes dynamically and thus there is no waste of resources.

StartApplication() of the existing source classes are commonly used to create a socket and send a packet but they also need to perform different things when start their application. Fig. 9 shows how to implement to do the same and different behaviors effectively by applying the template method pattern [8]. DoPreSomethingAtStart() and DoPoseSomethingAtStart() operations are inserted before and after the common behaviors that create a socket and send a packet. When extending the proposed Generic Application class, it is necessary to override these operations to do different behaviors for different application classes.

```
void StartApplication()
{
    DoPreSomethingAtStart();
    m_socket = socket Behavior->Set Socket(address, port);
    send Behavior->SendPacket(m_socket, ...);
    DoPostSomethingAtStart();
}
```

Figure 9. Implementation of StartApplication() of GenericApplication applying the template method pattern

IV. ARCHITECTURE COMPARISON

In this section, we compare the existing architecture of ns-3 and the proposed architecture. Table 8 show the result of comparison of the complexity of two architectures. The existing architecture should add a new application class by inheriting ns3::Application if a new traffic pattern is required. Thus we need (n+1) classes for n traffic patterns. However the proposed architecture needs only two classes: ns3::Application and GenericApplication. The different behaviors to set the socket and send a packet are modeled by implementing relevant interfaces. These kind of codes are already included in application classes of the form of the duplicated code in the existing architecture. In the proposed architecture, however, these codes are defined only in one place without duplication so that they can be re-usable for other traffic pattern and thus we do not count them.

Table 8. Comparison of the complexity

Kinds of application classes	Existing architecture	Proposed architecture
3	4	2
5	5	2
10	11	2

Table 9 show the comparison of characteristics of the existing architecture and the proposed one. In the existing architecture, code can be duplicated especially when implementing the similar operation such as setting a socket and sending a packet, but there is no code duplication in the proposed architecture. And also the combination of setting a socket and sending a packet is possible to support a new traffic pattern in the proposed architecture but it is not possible in the existing architecture.

Table 9. Comparison of characteristics

Architecture \ Characteristic	Code duplication	Combination
Existing architecture	O	X
Proposed architecture	X	O

In addition, the map collection of properties can store different attributes in the variability analysis dynamically, which makes the proposed architecture more flexible without any modification of codes to add a new attributes. In the existing architecture, a new attribute makes the programmer to modify the class structure and thus the regression testing is necessary to verify that there is no defect in the modified code and its relevant code.

V. CONCLUSION

Ns-3 is the discrete event network simulator developed using the object-oriented programming language C++. It is very complex in terms of the software architecture so that it is not easy to add a new class or modify existing classes. In this paper, we have focused on the architecture of classes related to the network traffic generation which is the most widely used classes to evaluate the network performance. The existing architecture inherits ns3::Application to make a new class for each traffic pattern, which shows the many duplicated code. In this paper, we have proposed a new application architecture by performing the commonality/variability analysis of both attributes and operations of the existing application classes to remove this problem. We have applied the strategy pattern and the template method pattern to model the same and different behaviors to remove the duplicated code in the existing application classes. It is expected that we can develop more flexible application class by implementing relevant interfaces without any duplicated code.

ACKNOWLEDGEMENTS

This work was supported by Dong-Eui University Foundation Grant (2015AA022).

REFERENCES

[1]. Ns-3 homepage, http://www.nsnam.org/.
[2]. Ns-2 homepage, http://nsnam.isi.edu/nsnam/index.php/Main_Page.
[3]. Enterprise Architect – UML Design Tools and UML CASE Tools for Software Development, http://www.sparxsystems.com.au/products/ea/index.html.
[4]. K. Fall and K. Varadhan (Ed.), *The ns Manual*, Nov. 4. 2011.
[5]. W. Jiang and H. Schulzrinne, Analysis of On-Off Patterns in VoIP and Their Effect on Voice Traffic Aggregation, *Proc. the 9th Int'l Conf. on Computer Communications and Networks*, Oct.16-18, 2000, pp.82-87.
[6]. Y. Liang, X. Liu, M. Zhou, Y. Liu and B. Shan, "A Robust Voice Activity Detector Based on Weibull and Gaussian Mixture Distribution," *2010 Second Int'l Conf. on Signal Processing Systems (ICSPS)*, vol.2, pp.26-30, 2010.
[7]. B.D. McLaughlin, G. Pollice and D. West, *Head-First Object-Oriented Analysis and Design* (O'Reilly Media Inc., 2007).
[8]. E. Freeman and E. Freeman, *Head First Design Patterns* (O'Reilly Media Inc., 2004).
[9]. E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software* (Addison-Wesley Professional, 1994).
[10]. R. Stewart (Ed.), *Stream Control Transmission Protocol*, RFC 4960, Sept. 2007. (Proposed Standard)
[11]. Map – C++ Reference, http://www.cplusplus.com/reference/map/map/.
[12]. Map (Java Platform SE 8), https://docs.oracle.com/javase/8/docs/api/java/util/Map.html.

Jong Min Lee received the B.S. degree in computer engineering from Kyungpook National University, Korea, in 1992, and the M.S. and the Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology (KAIST) in 1994 and 2000, respectively. From Sept. 1999 to Feb. 2002, he worked for Samsung Electronics as a senior engineer. Since 2002, he has been a faculty member of the Department of Computer Software Engineering, Dong-Eui University, Busan, Korea. From Feb. 2005 to Feb. 2006, he visited the University of California, Santa Cruz as a research associate. From Feb. 2012 to Feb. 2013, he was a visiting scholar of the Department of Computer Science at The University of Alabama, Tuscaloosa, AL. Since 2015, he has been a software process assessor of quality certification of Software Process in NIPA Software Engineering Center. His research interests include routing in ad hoc networks and sensor networks, mobile computing, and parallel computing.