A Traffic Generator for NEAT

An Extension of the D-ITG Traffic Generator for the NEAT Transport System

Carl Philip Matsson, Thodsapon Kaewprathum, Adam Larsson
Examiner: Karl-Johan Grinnemo

Faculty of Health, Science and Technology

Computer Science

A Traffic Generator for NEAT

An Extension of the D-ITG Traffic Generator for the NEAT Transport System

Carl Philip Matsson Thodsapon Kaewprathum

Adam Larsson

Karlstad University

13th January 2017

Examiner: Karl-Johan Grinnemo

Abstract

Ossification is one of the most serious problems facing the future of the Internet. The most commonly used protocols today can not keep up with the increasing demand for better network performance of the next generation of applications. New transport protocols have a hard time replacing the outdated ones. Even well-know protocols such as SCTP and DCCP have failed to become anything but niche protocols on the Internet. To solve this problem, a new transport system, NEAT, is currently under development. There is limited opportunities to evaluate NEAT in practice. Before this project, there was no traffic generator that supports NEAT which would help with evaluation during the development process. This project implements the NEAT transport system onto D-ITG, an existing open source traffic generator. D-ITG has many features that can be of use when evaluating NEAT, such as advanced logging and simulation of many different applications traffic signatures.

Contents

1 Introduction

The most common transport protocols used today are TCP and UDP. This has been the case ever since the Internet started gaining widespread popularity, and is one of the reasons middleboxes, such as policy enhancing proxies and firewalls, are almost exclusively optimized for these protocols (Nabi et al.)[7]. These middleboxes might block or change the content of, what in their view is, unknown types of traffic, making it hard for new innovative protocols to be deployed (Honda et al.)[6]. Today, this is becoming a problem due to the increase in demand for higher bandwidth and smaller delays needed to run modern applications since TCP and UDP are having a hard time meeting these requirements (Nabi et al.)[7].

To try and solve this problem, the New Evolutive API and Transport-Layer Architecture for the Internet (NEAT) project is focusing on developing an API and transport framework that separates applications and the protocols they use. This makes it easy to add new protocols as the applications requirements change. The transport system of NEAT is deployable with its design that is independent on the operating system and certain network technologies. The flexibility of NEAT transport system allows the new required protocols and mechanisms to be added which also allows applications to benefit from the new features as NEAT infrastructure evolves (Grinnemo et al.)[5].

By using NEAT, the developers of applications would also be able to specify services required by their applications, such as high bandwidth, congestion control or low delay, instead of actively selecting what specific transport protocol to use. This simplifies their implementation and ensures that they will get the most suitable protocol based on the needs of their application.

The work on NEAT is still ongoing, and there is a need to be able to evaluate the performance of NEAT in practice. This report aims to describe the author's attempt at implementing NEAT in an existing, open source, traffic generator. The traffic generator used in this project is the Distributed Internet Traffic Generator (D-ITG) [1]. D-ITG is a completely free and open source application published under the GNU license. D-ITG is able to generate IPv4/IPv6 data according to different distributions, or emulating different applications, and have extensive logging functionality, making it an ideal candidate for NEAT developers in their testing of the NEAT library.

2 Background

Section 2.1 will provide the background for the traffic generator used in the project, the section includes the basic explanation of the capabilities and functionality of different components of the traffic generator. Section 2.2 includes the background for the enhanced API and transport layer for the Internet. The section will briefly explain the issue with today's Internet transport layer and the proposed solution to the issue. Section 3 involves the original design-architecture of the traffic generator with an overview of the design, this section also provides a brief explanation of the functionality of different components in

the traffic generator. Next, in Section 3, the new architecture-design combination of the traffic generator and NEAT is provided with the explanation of the idea behind the design. The functionality of different components will also be described. Section 3.1 involves the client-side components such as the *ITGSend* and the NEAT client with the deeper explanation of the functionality and operating modes. This section also describes the extension of *ITGSend* with the NEAT client component and how they work together. Similar to Section 3.1, Section 3.2 describes the *ITGRecv* component in detail, and the functionality and the extension with NEAT server component. Section 4 provides the manual that explains the use of the NEAT option. Section 4.1 explains different NEAT properties within the JSON file. Section 4.2 provides information about the functionality of the client-side including the command line for data generation with different options, and the explanation of the script file used in multi-flow mode. In Section 4.3, the functionality of the server side is provided including the use of the *ITGRecv* component with the NEAT option. Section 4.4 describes the use of logging and how to create log files on both client side and server side, and how to analyze the log files using the *ITGDec* component. Finally in Section 5, the use cases of the project are provided with some examples from the results.

2.1 Distributed-Internet Traffic Generator

The Distributed Traffic Generator (D-ITG) is a traffic generator created and implemented by the University of Naples Federico II in Italy. D-ITG is an open-source software capable of generating both IPv4 and IPv6 traffic at the network, transport, and the application layer where it emulates several applications such as the Domain Name System (DNS), Voice-over-IP (VoIP) and Real-time Transport Protocol (RTP), and for this reason, it has become widely used. D-ITG is capable of generating traffic that accurately emulates the behavior of an arbitrary application by letting the user specify the sizes of individual packets and their inter-departure times. Several probability distributions are supported by D-ITG such as Exponential, Uniform, Normal, Poisson, Cauchy, Pareto, Gamma, and Weibull distributions and also several transport protocols including Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Stream Control Transmission Protocol (SCTP) and Datagram Congestion Control Protocol (DCCP). D-ITG also provides the possibility to generate multiple traffic flows.

D-ITG consists of two main components, *ITGSend* and *ITGRecv*, which are responsible for generating and receiving the traffic flows, respectively. *ITGSend* is able to generate and send multiple parallel traffic flows towards multiple *ITGRecv* instances, and *ITGRecv* is capable of receiving multiple parallel traffic flows generated from multiple *ITGSend* instances (Botta et al.)[4].

D-ITG can be operated in three different modes: single flow, multi flow and daemon mode. In single-flow mode, *ITGSend* generates a single traffic flow by reading the configuration of the flow from the command line, and by sending the generated flow towards a single *ITGRecv* instance. One thread manages the generated traffic flow and another thread controls the flow generation process. In multi-flow mode, however, *ITGSend* generates multiple traffic flows by reading

the configuration of each flow from a script file, and by sending each generated traffic flow toward one or more *ITGRecv* instances (Botta et al.)[4]. Each flow in the script file is represented by a line, and is characterized by different flow options including transport protocol, packet size, packets-per-second, duration, etc. Because there are multiple flows being generated, each flow in multi-flow mode has to be managed by a single thread. Another, separate thread manages the flow generation process. In daemon mode, *ITGRecv* will run as a daemon which listens to a UDP socket for instructions (Botta et al.)[4].

D-ITG is particularly useful with its *ITGLog* component. *ITGLog* provides the possibility for the user to log traffic information generated and received by *ITGSend* and *ITGRecv*, respectively. It receives and stores log information from multiple *ITGSend* and *ITGRecv* instances, and can log the traffic information both locally and remotely (Botta et al.)[4]. *ITGLog* can be operated on remote logging mode by using the remote log server which enables the logging operation to be operated on a remote log server. The log information in the log file includes the duration of experiment, total packets transferred, minimum delay, maximum delay, average delay, average jitter, the standard deviation of the delay, total bytes received, average bit-rate, average packet rate, packet dropped rate, average loss-burst size, number of duplicated packets, first sequence number, last sequence number, and loss events.

The *ITGLog* component logs all traffic information that is received from the *ITGRecv* component in a binary-encoded file. To be able to read this file, the user needs to decode it to a human readable format. The decoding process of the log file can be done by another D-ITG component called *ITGDec*. *ITGDec* is the D-ITG decoder component responsible for decoding and analyzing the encoded traffic information in log file

2.2 A New, Evolution API and Transport-Layer Architecture for the Internet

Because TCP and UDP are the two most widely used transport protocols today, introducing and deploying new transport protocols has become almost impossible. Even well-known protocols such as SCTP and DCCP have failed to become anything but niche protocols on the Internet. This has raised the question of whether the Internet transport layer is indeed ossified or not.

NEAT is believed to be a solution to the ossification issue. NEAT is an enhanced transport-layer API and transport system that allows applications to specify their requirements from the network services. NEAT allows the applications to select certain network services such as reliability, low delay and security dynamically. The selection of network services is based on application requirements, current network conditions or hardware capabilities. The overview of the NEAT design-architecture is shown in Figure 1.

The NEAT User Module is designed to be portable across different operating systems and network stacks, and is comprised of five groups of components in the form of the NEAT Framework, the NEAT Selection, the NEAT Policy, the NEAT Transport and the NEAT Signaling as shown in Figure 2. The NEAT Framework components provide a basic functionality that are needed in order

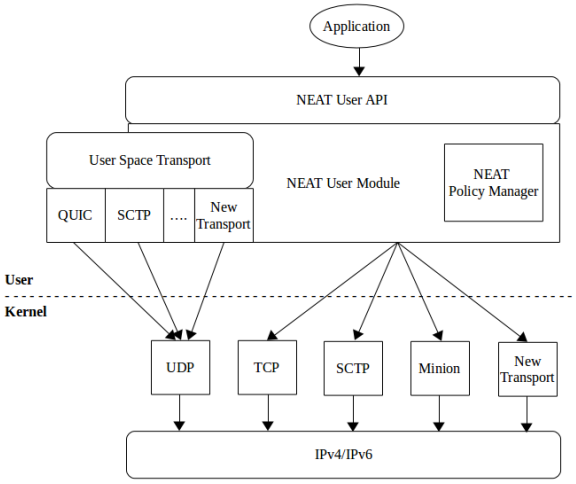

Figure 1: NEAT Design-Architecture

to use the NEAT System which define the structure of the NEAT User API and interfaces to the NEAT Logic which implements core mechanisms. The NEAT User API is used by the applications as the API to provide information about all the requirements for the desired transport services (Grinnemo et al.)[5].

The NEAT Selection components are responsible for choosing the appropriate transport services. The NEAT User API provides additional information that enables the NEAT System to make the stack aware of the differences between what is *desired* or *required* for each traffic flow. Based on the information provided by the NEAT User API and policies for service selection, the NEAT Policy components are able to identify the candidate transport solutions which is then tested by the NEAT Selection components, and only one appropriate transport solution is chosen and returned to the NEAT Logic.

The NEAT Policy components comprise of the Policy Information Base (PIB), the Characteristic Information base (CIB) and the Policy Manager where the PIB is the repository containing the collection of policies. Each policy is a set of rules that link a set of matching requirements to a set of preferred or mandatory transport characteristics, and the CIB is a repository that stores the information about available interfaces, supported protocols, network properties and current and previous connections between endpoints. The Policy Manager allows a set of rules to specify the transport protocol to be used for certain transport service. The NEAT transport components provide the functionality for instantiating the transport service for a certain NEAT flow, these components also provide a set of transport protocol including TCP, UDP and SCTP and other components that are necessary for realizing a transport service. The NEAT Selection components will be choosing the transport protocols and the NEAT Transport components will be configuring and managing the chosen transport protocols. The NEAT Signalling components provide the signal to complement the functions of the NEAT Transport components which could include the communication with middleboxes, support for failover, handover and other mechanisms.(Grinnemo et al.)[5]

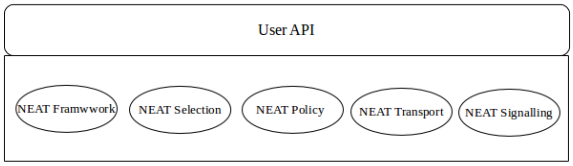

Figure 2: The NEAT User Module

3 Design

The design-architecture of D-ITG comprises five components in the form of *ITGSend*, *ITGRecv*, *ITGLog*, *ITGDec* and *ITGManager* that work together to generate, log, analyze and manage the traffic as shown in Figure 3. *ITGSend* is responsible for generating the traffic flow which is forwarded toward *ITGRecv*. *ITGRecv* is the component responsible for receiving the traffic flow. ITGSend is able to generate one or multiple parallel traffic flows toward one or multiple *ITGRecv* instances depending on its mode of operation. *ITGSend* is able to operate in three different modes: single-flow, multi-flow and daemon mode.

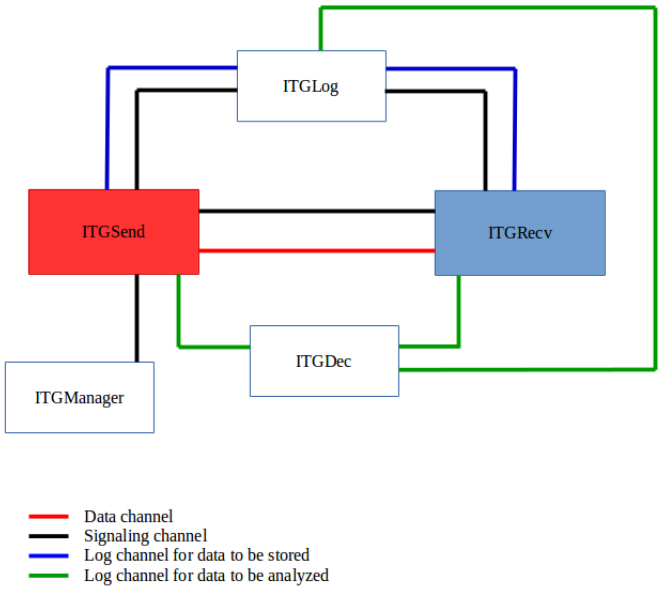

Figure 3: The design-architecture of D-ITG

Additionally, *ITGRecv* is capable of receiving multiple traffic flows generated by either one or multiple instances of *ITGSend*. A signaling channel is created between *ITGSend* and *ITGRecv* which is used to control and manage all the traffic flows being generated between them by using the Tunnel Setup Protocol (TSP) to signal the traffic generation process (Grinnemo et al.)[5]. Another feature of D-ITG is that both *ITGSend* and *ITGRecv* can optionally create log files which can be saved locally or remotely by sending the log files to *ITGLog*

which is the D-ITG log component responsible for receiving and storing log information. The log files contain information of every sent and received flow. *ITGDec* is the component responsible for decoding and analyzing the log files to get all the information of the traffic flows.

The *ITGRecv* component can be run as a daemon and may be completely configured and controlled by *ITGSend*. The same can be done to the *ITGSend* component.

The *ITGManager* component can be used to remotely control the *ITGSend* component. This feature is a very useful for large scale distributed experiment where the user has the complete control of the experiment from a single vantage point.

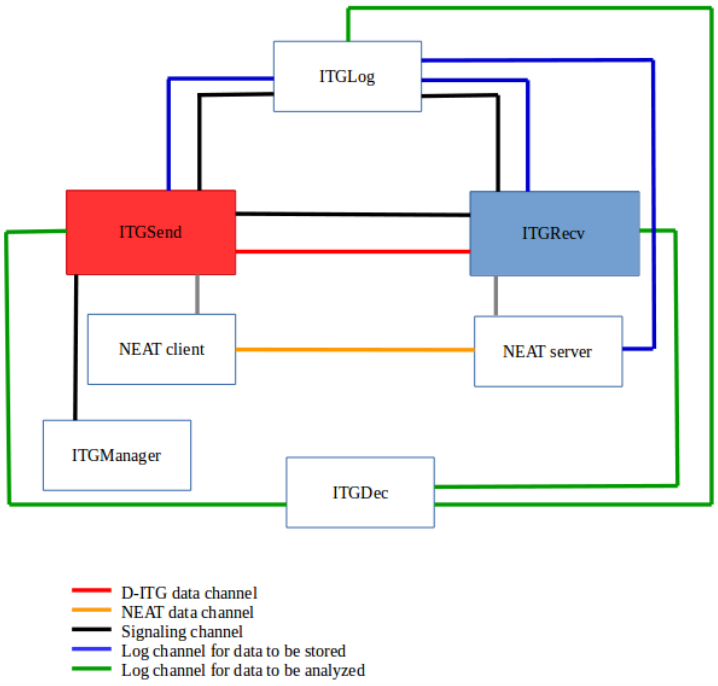

Figure 4: The design-architecture of D-ITG extended with NEAT

With the addition of this project, D-ITG has been extended with two new components, the NEAT client and the NEAT server, as can be seen in Figure 4. There has also been some minor functionality added to existing components such as logging.

The idea behind the design were to keep as much of the D-ITG code base intact and have NEAT as separate components while still keeping all the functionality provided by D-ITG, making upgrading to new versions of D-ITG or NEAT simpler.

The system works as follows: Each time a new NEAT flow is requested by the user, a new NEAT client is created on a separate thread. The main D-ITG application waits until a connection between the NEAT client and server is

established before it continues processing. Because of the way D-ITG is implemented and the fact that one of the goals were to leave as much of the original code base intact as possible, a connection between *ITGSend* and *ITGRecv* is also required for the program to continue. Thus a "dead" TCP connection is established between *ITGSend* and *ITGRecv*. When the connection is established, *ITGSend* starts generating data. The data is forwarded to the NEAT client, which in turn transmits the data to the NEAT server. All logging that takes place at the sender, use the default D-ITG implementation in *ITGSend*. It also logs what protocol were used for the connection.

The server side logging with the default D-ITG implementation is usually done within the socket code. Since NEAT does not use these sockets for its connection, the server-side logging for NEAT flows is implemented within the NEAT server. The NEAT server stores a buffer of information about the packets, and when the buffer is full, forwards them to *ITGLog* for processing and printing to the log files.

3.1 ITGSend

ITGSend is the D-ITG component responsible for generating the traffic. It is capable of generating both IPv4 and IPv6 traffic. *ITGSend* is also able to generate multiple parallel traffic flows toward one or multiple *ITGRecv* instances when operating in multi-flow mode.

As mentioned earlier, *ITGSend* can be operated in three different modes, single-flow, multi-flow and daemon mode. When operating in single-flow mode, *ITGSend* generates a single traffic flow towards a single *ITGRecv* instance by reading the configuration such as destination address, packet rate, packet size, duration of the traffic generation and protocol type from the command line. For this flow, one thread is dedicated to manage the flow itself, a separate thread is managing the connection set up and the traffic flow generation process. It does all of this by communicating with the *ITGRecv* component via the signaling channel.

Meanwhile, in multi-flow mode, instead of reading the flow configuration directly from the command line, the script file is used for the generation of traffic flows. The script file contains the configuration for each traffic flow which specifies the different characteristic of the flow and each line in the script file represents a single traffic flow. *ITGSend* then reads this script file from the command line to generate the traffic flows which is later sent toward one or multiple instances of *ITGRecv*. Each flow in multi-flow mode is managed by a single thread with another separate thread manages and controls the traffic flow generation process.

In daemon mode, *ITGSend* can be remotely controlled by *ITGapi* which is the C++ API that allows the traffic generation process to be remotely controlled. *ITGSend* acts as a daemon which listens to the UDP socket for traffic generation request when operating in deamon mode.

3.1.1 NEAT Client

ITGSend has been extended with a new component, a NEAT client. When using the NEAT protocol, instead of forwarding the data directly to *ITGRecv*, it is forwarded through the NEAT client that proceseses this data and transmits it to the NEAT server. The NEAT client is a part of *ITGSend* but functions as a separate component. For each NEAT flow that is generated by *ITGSend*, a new NEAT client thread is started with its own event loop.
The NEAT client starts by parsing the neat option string, forwarded by *ITGSend*. The resulting values of the parsed input are added to a container in which each index contains a unique flow and its properties. An ID for each flow is generated by D-ITG and is re-purposed to keep track of the location of each NEAT flow and its parameters in the container. This makes it easy to operate on the NEAT flows from outside of the NEAT client component.

The NEAT client creates a new flow, which is a communication link to which data is read and written. NEAT determines which transport protocols can support the properties of the ones defined in the JSON property file. NEAT tries to establish a connection, between the NEAT client and NEAT server, using one protocol at a time until a connection has successfully been established [3]. Data is then generated by *ITGSend* and passed to the NEAT client, with the flow id and is transmitted to the NEAT server via the flow corresponding to the id.

3.2 ITGRecv

ITGRecv is a D-ITG receiving component responsible for receiving multiple parallel flows from the generating and sending component, *ITGSend*. *ITGRecv* is capable of receiving traffic flows generated by one or multiple instances of *ITGSend*. A new thread is created every time *ITGRecv* receives a request; the created thread then performs all the related operations to the new request such as receiving the packets of the flow.

3.2.1 NEAT Server

The NEAT server is a part of *ITGRecv* but functions as a separate component. It runs in its own thread, and only deals with NEAT flows and NEAT-related logging functionality.

The NEAT server starts by parsing the NEAT option string that is forwarded as parameter by *ITGRecv* when the NEAT server thread is created.

For each NEAT-client request, the NEAT server creates a new flow which is used to receive data from the NEAT client. The NEAT server starts the event loop, which is the program loop that polls the flows and all events that occur. When an event is triggered, the assigned callback function code will be executed.
The callbacks that are used in the NEAT server are, *on_connected* which is triggered when an incoming connecting has been established, *on_readable* which is triggered when there is data that can be read from the flow and *on_error* which is triggered when an error occurs. When the *on_connected* callback is made, the

NEAT server establishes what code should execute for the *on_readable* callback. This makes the server able to retrieve and handle packets.
The *on_readable* callback is made when a packet sent from the NEAT client is readable on the flow. Every time this callback is made, the information about the packet, such as sequence number and packet size, is added to a container that stores information about each packet. After all packets have been sent by the NEAT client or the container has reached its max capacity, this container is written to the log file.

4 Manual

For installation and compilation of the application, see the github manual.

The NEAT transport system implementation is built on top of D-ITG, and is to a large extent separated from main application. Thus, it is possible to use all available features that D-ITG currently provides (Botta et al.)[4].

The additional option that is available with this project is the NEAT option (-NO). The options specified after this tag are NEAT specific, and will only modify the behavior of NEAT. The NEAT-options tag should always be followed by quotation marks, and all NEAT options should be defined within these quotation marks, even when using a single option.

`-NO "⟨option⟩"`

`-NO "⟨option⟩ ⟨option⟩ ... ⟨option n⟩"`

NEAT has been incorporated into D-ITG as a separate protocol under the default D-ITG protocol tag (-T).

```
-T NEAT
```

4.1 NEAT Properties

At the time of this writing, the only way to customize properties for NEAT is through a JSON file. This file allows the user to customize properties such as what transport protocols are available and security properties.

At the root of the project there is a file called *custom.json* that contains a simple example of a property file.

```
{
    "transport": [
        {
            "value": "TCP",
            "precedence": 1
        },
        {
            "value": "SCTP",
            "precedence": 1
        },
        {
            "value": "UDP",
            "precedence": 1
        }
    ]
}
```

The transport property consists of an array containing the transport protocols that the user want NEAT to be able to select from [2].

4.2 Client

The client will generate data based on the D-ITG options the user provides. The client can either start a single instance from the command line, or multiple instances by using a user provided file containing information about the flows.

4.2.1 Command Line

To start one client from the command line, use the following form.

`$ ./ITGSend` ⟨*option*⟩ ⟨*option*⟩ ... ⟨*option n*⟩

For example, to start a flow with a constant payload size of 100 bytes, constant packet rate of 10 packets/s, using NEAT for 15 seconds with the NEAT properties defined in *custom.json*:

```
$ ./ITGSend -a 192.168.0.1 -c 100 -C 10 -t 15000 -T NEAT -NO "-P ../custom.json"
```

4.2.2 File

To start multiple flows at once, a script file has to be used. The options for each flow is provided on a single line.

For example, creating a file that is defining three flows using TCP, NEAT and UDP:

```
script_file:
-------------------------------------------------------------------------
| -a 192.168.0.1 -c 100 -C 10  -t 15000 -T TCP                          |
| -a 192.168.0.1 -c 100 -C 10  -t 15000 -T NEAT -NO "-P ../custom.json" |
| -a 192.168.0.1 -c 100 -C 10  -t 15000 -T UDP                          |
|_______________________________________________________________________|
```

To execute the flows:

```
$ ./ITGSend script_file
```

4.3 Server

The server receives and logs the incoming flows. To start the server in its default state with the default NEAT properties defined in the code, use:

```
$ ./ITGRecv
```

To customize what properties NEAT use, specify the NEAT properties after the NEAT options tag (-NO) and supply a JSON file:

```
$ ./ITGRecv -NO "-P ../custom.json"
```

4.4 Logging

Logging has been extended with additional useful information such as what protocol NEAT decided to use. Logging works the same as in D-ITG. a user issues the log sender tag (-l) to log the client side information and the receiver tag (-x) to log at the server side.

```
$ ./ITGSend -a 192.168.0.1 -c 100 -C 10 -t 15000 -T TCP -l sender.log -x receiver.log
$ ./ITGSend script_file -l sender.log -x receiver.log
```

4.4.1 Decoding Logs

The logs can be decoded and read by using:

```
$ ./ITGDec ⟨logfile⟩
```

5 Use Cases

In this chapter some possible use cases are defined. One of the use cases starts a single flow from the command line and the other use case shows how multiple flows can be implemented and executed from a script file.

In addition to the standard logs from D-ITG, a new protocol row has been added. This row allows the user to observe if the flow is using NEAT, and what protocol NEAT decided to implement.

Due to current limitations of NEAT, there is no way to retrieve the IP address or port of a flow. The IP addresses in the senders logfile are supplied by *ITGSend* and the NEAT server currently always use port 8080. Port number 0 implicates that the port number was not able to be retrieved from NEAT.

To run a single NEAT flow from the command line, the following is entered into the command line:

```
./ITGSend -a localhost -C 10 -c 30 -T NEAT
   -t 2000  -NO "-P ../custom.json" -l sender.log -x receiver.log
```

The properties file is defined to allow NEAT to choose between TCP or UDP.

This produces the following result:

sender.log:

```
Flow number: 1
Protocol: NEAT -> TCP
From: 127.0.0.1:0
To:   127.0.0.1:8080

Total time                =       1.910229 s
Total packets             =             20
Minimum delay             =       0.000000 s
Maximum delay             =       0.000000 s
Average delay             =       0.000000 s
Average jitter            =       0.000000 s
Delay standard deviation  =       0.000000 s
Bytes received            =            600
Average bitrate           =       2.512788 Kbit/s
Average packet rate       =      10.469949 pkt/s
Packets dropped           =              0 (0.00 %)
Average loss-burst size   =       0.000000 pkt
Packets duplicated        =              0
First sequence number     =              1
Last sequence number      =             20
Loss Events               =              0
```

receiver.log:

```
Flow number: 1
Protocol: NEAT -> TCP
From: <unknown>:0
To:   <unknown>:8080

Total time                =       1.910324 s
Total packets             =             20
Minimum delay             =       0.000163 s
Maximum delay             =       0.000579 s
Average delay             =       0.000290 s
Average jitter            =       0.000117 s
Delay standard deviation  =       0.000107 s
Bytes received            =            600
Average bitrate           =       2.512663 Kbit/s
Average packet rate       =      10.469428 pkt/s
Packets dropped           =              0 (0.00 %)
Average loss-burst size   =       0.000000 pkt
Packets duplicated        =              0
First sequence number     =              1
Last sequence number      =             20
Loss Events               =              0
```

In addition to running a single flow from the command line, it is also possible to run multiple flows simultaneously by defining the flows in a file.

```
script_file:
-a localhost -C 10 -c 64 -T TCP -t 15000
-a localhost -C 10 -c 64 -T NEAT -t 15000  -NO "-P ../custom.json"
```

With the flows defined in the file, *ITGSend* can take this file as a parameter and create the flows automatically.

```
./ITGSend script_file -l sender.log -x receiver.log
```

The resulting log files *sender.log* and *receiver.log* may look like this:

sender.log:

```
Flow number: 1
Protocol: TCP
From: 127.0.0.1:48254
To:   127.0.0.1:8999
```

Total time	=	14.991265	s
Total packets	=	150	
Minimum delay	=	0.000000	s
Maximum delay	=	0.000000	s
Average delay	=	0.000000	s
Average jitter	=	0.000000	s
Delay standard deviation	=	0.000000	s
Bytes received	=	9600	
Average bitrate	=	5.122983	Kbit/s
Average packet rate	=	10.005827	pkt/s
Packets dropped	=	0	(0.00 %)
Average loss-burst size	=	0.000000	pkt
Packets duplicated	=	0	
First sequence number	=	1	
Last sequence number	=	150	
Loss Events	=	0	

receiver.log:

```
Flow number: 1
Protocol: TCP
From: 127.0.0.1:48254
To:   127.0.0.1:8999
```

Total time	=	14.991430	s
Total packets	=	150	
Minimum delay	=	0.000019	s
Maximum delay	=	0.000715	s
Average delay	=	0.000268	s
Average jitter	=	0.000117	s
Delay standard deviation	=	0.000127	s
Bytes received	=	9600	
Average bitrate	=	5.122927	Kbit/s
Average packet rate	=	10.005717	pkt/s
Packets dropped	=	0	(0.00 %)
Average loss-burst size	=	0.000000	pkt
Packets duplicated	=	0	
First sequence number	=	1	
Last sequence number	=	150	
Loss Events	=	0	

sender.log:

```
Flow number: 2
Protocol: NEAT -> TCP
From: 127.0.0.1:0
To:   127.0.0.1:8080
```

Total time	=	14.991371	s
Total packets	=	150	
Minimum delay	=	0.000000	s
Maximum delay	=	0.000000	s
Average delay	=	0.000000	s
Average jitter	=	0.000000	s
Delay standard deviation	=	0.000000	s
Bytes received	=	9600	
Average bitrate	=	5.122947	Kbit/s
Average packet rate	=	10.005756	pkt/s
Packets dropped	=	0	(0.00 %)
Average loss-burst size	=	0.000000	pkt
Packets duplicated	=	0	
First sequence number	=	1	
Last sequence number	=	150	
Loss Events	=	0	

receiver.log:

```
Flow number: 2
Protocol: NEAT -> TCP
From: <unknown>:0
To:   <unknown>:8080
```

Total time	=	14.991341	s
Total packets	=	150	
Minimum delay	=	0.000078	s
Maximum delay	=	0.001976	s
Average delay	=	0.000331	s
Average jitter	=	0.000184	s
Delay standard deviation	=	0.000238	s
Bytes received	=	9600	
Average bitrate	=	5.122957	Kbit/s
Average packet rate	=	10.005776	pkt/s
Packets dropped	=	0	(0.00 %)
Average loss-burst size	=	0.000000	pkt
Packets duplicated	=	0	
First sequence number	=	1	
Last sequence number	=	150	
Loss Events	=	0	

sender.log:

```
------------------------------------------------------------
***************  TOTAL RESULTS    ***************
------------------------------------------------------------
Number of flows          =            2
Total time               =     15.997221 s
Total packets            =          300
Minimum delay            =      0.000000 s
Maximum delay            =      0.000000 s
Average delay            =      0.000000 s
Average jitter           =      0.000000 s
Delay standard deviation =      0.000000 s
Bytes received           =         19200
Average bitrate          =      9.601668 Kbit/s
Average packet rate      =     18.753257 pkt/s
Packets dropped          =            0 (0.00 %)
Average loss-burst size  =            0 pkt
Error lines              =            0
------------------------------------------------------------
```

receiver.log:

```
------------------------------------------------------------
***************  TOTAL RESULTS    ***************
------------------------------------------------------------
Number of flows          =            2
Total time               =     15.997470 s
Total packets            =          300
Minimum delay            =      0.000019 s
Maximum delay            =      0.001976 s
Average delay            =      0.000300 s
Average jitter           =      0.000149 s
Delay standard deviation =      0.000194 s
Bytes received           =         19200
Average bitrate          =      9.601518 Kbit/s
Average packet rate      =     18.752965 pkt/s
Packets dropped          =            0 (0.00 %)
Average loss-burst size  =            0 pkt
Error lines              =            0
------------------------------------------------------------
```

References

[1] D-itg, distributed internet traffic generator. http://www.grid.unina.it/software/ITG/. [Online; Accessed 06-December-2016].

[2] Neat properties. http://neat.readthedocs.io/en/latest/properties.html, . [Online; Accessed 07-December-2016].

[3] Neat tutorial. http://neat.readthedocs.io/en/latest/tutorial.html, . [Online; Accessed 15-December-2016].

[4] A. Botta, W. Donato, A. Dainotti, S. Avallone, and A. Pescap. D-itg manual. http://traffic.comics.unina.it/software/ITG/manual/. [Online; Accessed 07-December-2016].

[5] K-J Grinnemo, T Jones, G Fairhurst, D Ros, A Brunstrom, and P Hurtig. Neat a new, evolutive api and transport-layer architecture for the internet, June 2016.

[6] M. Honda, T. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend tcp? http://dl.acm.org/citation.cfm?id=2068834, November 2011. [Online; Accessed 01-December-2016].

[7] Z. Nabi, T. Moncaster, A. Madhavapeddy, S. Hand, and J. Crowcroft. Evolving tcp: how hard can it be? http://dl.acm.org/citation.cfm?id=2413270, December 2012. [Online; Accessed 01-December-2016].

ISBN 978-91-7063-742-1

Faculty of Health, Science and Technology

PROJECT REPORT DVAE08/2017/1 | January 2017