SJBS (M), (2016b), 34: 85-102 ISSN: 7615-1858

Parallel Path Computation Dijkstra Simulation(PCE)

Rawya Elmahi Gobara Salliem[1]

Abstract:

In this paper, Path Computational Element (PCE) has been extensively studied under European Union project called PACE. However, much work needs to be done when it comes to Multi-layer PCE rchitecture. The computational algorithm part of PCE has been researched extensively , here we propose a parallel computational algorithm for solving this problem. PCE uses dijkstra's algorithm for finding the shortest path of a one layer topological PCE architecture. In Multi-layer PCE architecture, the dijkstra algorithm has to be modified to accommodate the different layers of the topological hierarchy. To create the Parallel path computation algorithm, we use the lambda function of the Python code of hierarchical PCE. The lambda function is based on map-reduce which maps the different parallel tasks of finding shortest path of the hierarchy and reduce the result by choosing the shortest path from the different parallel computed tasks. By introducing the parallel computational algorithm for multi-layer PCE, the complexity of the computational problem is resolved and the performance is increased.

Keywords: MPLS-TE, CSPF, Lambda, Path computation Element (pce), supercomputer .

[1] Department of Research and development, Institute of computer center, Sudan University of Science and Technology Sudan.

1. Introduction:

The emergence of dynamic connection-oriented services has led to need for increased telligence to support on-demand service provisioning[7] new network architectures[1] are enabling edge systems to directly control their network resources through specialized signaling. These new capabilities can be accessed through policy- based processes by independent organizations, individuals and even applications. So path computational element (PCE) [2] has been extensively studied under european union project called path usually a huge amounts of requests occur at any moment, it needs to quickly find the solution.In this research, we have created multiple layers rather than one layer, which are parallel layers that divide We used Python instead of Java to illustrate the output and use the Dijitsktra's algorithm[6] to access the shortest path in the network To create the Parallel path computation algorithm, we use the lambda function of the Python code of hierarchical PCE. The lambda function is based on map-reduce which maps the different parallel tasks of finding shortest path of the hierarchy[3] and reduce the result by choosing the shortest path from the different parallel computed tasks. By introducing the parallel omputational algorithm for multi-layer PCE, the complexity of the computational problem is resolved and the performance is increased[8] .

2-Materials and methods of research:

The researcher relied on a set of tools in collecting the information and data imposed by the nature of the research and the tools are as follows:

(a) The direct interview benefited from the researcher in monitoring the new program outputs through follow-up.

(b) The importance of the interview as a research tool in that it is suitable for research that deals with specific research cases on specific models and sectors, where the study seeks to discover access to the nearest point in the network and the least time and cost possible.

Advanced Computer Training Institute: is one of the main tools in the open source program for data collection, analysis and programming within the framework allowed with open source programs

(a) Human Framework: The research community under study is a number of programming experts, network engineers with network expertise and volunteer experts in open source software initiatives.
(b) Tentative Framework:The period from December 2015 to December 2019.
(c) Spatial Framework: Sudan - Khartoum State

2.2 Python:

Python is a high-level scripting language that offers an interactive programming environment Programming languages generally have the following common ingredients: variables, operators, iterators, conditional statements, functions (built-in and user defined) and higher-order data structures. We will look at these in Python and highlight qualities unique to this language.

2.3 Dijkstra's Algorithm and Best-First-Search:

Dijkstra's Algorithm works by visiting vertices in the graph starting with the object's starting point. It then repeatedly examines the closest not-yet-examined vertex, adding its vertices to the set of vertices to be examined. It expands outwards from the starting point until it reaches the goal. Dijkstra's Algorithm is guaranteed to find a shortest path from the starting point to the goal, as long as none of the edges have a negative cost. (I write "a shortest path" because there are often multiple equivalently-short paths.) In the following diagram, the pink square is the starting point, the blue square is the goal, and the teal areas show what areas Dijkstra's Algorithm scanned. The lightest teal areas are those farthest from the starting point, and thus form the "frontier" of exploration

2.4 Mapreduce:

The mapreduce parallel programming model has become extremely popular in the big data community. Many bigdata

workloads can benefit from the enhanced performance offered by supercomputers. Lmapreduce provides the familiar map-reduce parallel programming model to big data usersrunningon a supercomputer. Lmapreduce dramatically simplifies map-reduce programming by providing simple parallel programming capability in one line of code. LMapReduce supports all programming languages and many schedulers. Lmapreduce can work with any application without the need to modify the application. Furthermore, Lmapreduce can overcome scaling limits in the map-reduce parallel programming model via options that allow the user to switch to the more efficient single-program-multiple-data (SPMD) parallel programming model. These features allow users to reduce the computational overhead by more than 10x ompared to standard map-reduce for certain applications. Lmapreduce is widely used by hundreds of users at mit. Currently Lmaprduce works with several schedulers such as slurm, gridengine and lsf.

2.5 Supercomputer:

A supercomputer is a computer at the frontline of current processing capacity, particularly speed of calculation". Or –A computer that enables research that cannot be practically performed on a esktop computer.

3. Background:

The basic PCE architecture is shown in Fig. 1. In a nutshell, the PCE is a server which can perform constrained path computation using topology information stored in its Traffic Engineering Database (TED). The TED contains information of the network topology including QoS parameters such as available capacity, link delay, etc

A. PCE Protocol (PCEP):

The PCE protocol (PCEP) is an integral part of the PCE which enables communication between two PCE peers. The protocol specifications in [2] define seven unique message types: The Open and Close messages are used to initialize and close the connection, the path computation requests are sent in a Path Computation Request message to which the response is sent in

the Path Computation Response Message and the Keepalive, Notification and Error are used in order to convey additional information to remote peers.A typical PCEP Fig (3.5) message format consists of a standard header and body and information inside the message body is encapsulated in the form of PCEP objects which use Type Length Value type representations. Each message contains some mandatory objects and some optional objects based on the type of information exchange required: for example, a Keepalive message is used only to check if a remote peer is still active and does not have any mandatory objects while a complex message such as the Path Computation Request message must contain an RequestParameters object containing the Request ID and an EndPoints object which defines the end- points for a path computation request and can additionally contain objects such as the Bandwidth and Metric objects which provide constraints on the path computation request.

4. Object-Oriented PCE Protocol Mapping:

The PCE protocol is a critical component in the PCE architecture Fig (3.5) as it is likely to be frequently updated to provide additional functionality in the PCE architecture or address protocol extensions. In our architecture, the TLV like objects of the PCE protocol are mapped to an object-oriented class hierarchy for internal usage. Therefore, PCEP messages are represented as objects inside the different modules making it easier to define internal logic based on them.The use of an object-oriented hierarchy makes inter-module PCEP message exchange easier when incorporating changes to the protocol, such as inclusion of new PCEP messages or objects. The use of the object-oriented hierarchy also means that all PCEP messages, regardless of message and object types are represented as a PCEPMessage object, thus ensuring that simple changes in the implementation do not lead to changes in all modules.In the architecture, the PCE protocol is mapped into a simple object-oriented class hierarchy as shown in Fig. 5. Each PCEP message is represented as a single object, which consists of a header and a messageFrame. While the header remains unchanged for different message type, a unique PCEP

message type is implementedas an independent class which implements the message frame interface. The interface contains functions to add PCEP objects and provides an interface to check the validity of the message where implementers can incorporate conditions to check for necessary objects. For example, for a regular PCEP Path Computation Request, we incorporate checks to ensure that a *RequestParameters* object

The computation module is responsible for processing path computation requests coming to the PCE server. The path computation requests inside the PCEP protocol are identified as point-to-point path requests along with a set of constraints. The computation module contains implementations of algorithms for supporting path computation and mechanisms to execute policy decisions to be imposed on path computation requests **Fig (2.5)** . In our architecture, we do not restrict the choice of TED used by the computation module for path computation and the developer can integrate the computation algorithms with a TED of choice. As a consequence, we also do not inherently support any specific mechanism to handle topology updates during path computation.

The performance of the computation module is highly critical to the operation of the PCE. In specialized networks such as WDM, computation algorithms can be complex requiring large processing resources. In our architecture, developers can address this issue by using more than one computation modules to balance the implementation complexity. In addition, it is not only possible to optimize the processing resources used inside the computation modules but also across all PCE modules, by reducing the processing resources available for use by the network and the session management modules. This will be discussed in more details in the performance results sections, we present the implementation details ofthe PCE emulator and its modules. The implementation was designed to support a large number of concurrent sessions, which dictated the design choices for the network and the session management modules. For implementation, we used Java to ensure operating system independence and the imple- mentation is compatible with Java compilers and eclips.

Below are the detailed steps used in Dijkstra's algorithm[6] to find the shortest path from a single source vertex to all other vertices in the given graph.Algorithm

(1) Create a set sptSet (shortest path tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.
(2) Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.
(3) While sptSet doesn't include all vertices
    (a) Pick a vertex u which is not there in sptSet and has minimum distance value.
    (b) Include u to sptSet.
    (c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

5. Appendix A ILP Formulation:

In this appendix, we present the formulation used for the ILP to serve batch lightpath requests. The graph was represented as a directed graph tt(V, E), with nodes vi $\in$ V and edges eij $\in$ E. The set of wavelengths was given by Tand the availability of a wavelength λt on edge eij was given by a boolean indicator ct . The batch of lightpath demands was defined as a set D and each demand dx $\in$ D has a source S(dx) and a destination D(dx) node, and was assumed to use only one wavelength. We defined an indicator variable L(dx) to indicate if a request was provisioned, and defined a boolean routing vari- able for demand dx as rt (dx), indicating if the lightpath used for demand dx used the wavelength λt on the link eij

Based on these parameters, the constraints for provisioning the lightpaths are given as:

$$\forall d_x \in D : L(d_x) \leq 1 \quad (1)$$

$$\forall d_x \in D, e_{ij} \in E, t \in T : r_{ij}^{t}(d_x) \leq L(d_x) \quad (2)$$

$$\forall d_x \in D, v_s = S(d_x) : \sum_{t \in T} \sum_{j : e_{sj} \in E} r_{sj}^{t}(d_x) = L(d_x) \quad (3)$$

$$\forall d_x \in D, v_s = D(d_x) : \sum_{t \in T} \sum_{j : e_{js} \in E} r_{js}^{t}(d_x) = L(d_x) \quad (4)$$

$$\forall d_x \in D, t \in T, j : v_j \neq D(d_x), S(d_x) : \sum_{i : e_{ij} \in E} r_{ij}(d_x) = \sum_{k : e_{jk} \in E} r_{jk}(d_x) \quad (5)$$

$$\forall d_x \in D, t \in T, : \sum_{e_{ij} \in E} r_{ij}^{t}(d_x) \leq HopLimit \quad (6)$$

$$\forall e_{ij} \in E, t \in T : \sum_{d_x \in D} r_{ij}^{t}(d_x) \leq c_{ij}^{t} \quad (7)$$

In these constraints, (1) is used to indicate if a demand is provisioned or not, and (2) ensures that routing variables are set to 0 if a demand is not provisioned. Constraints (3) and (4) ensure that at max. one wavelength is selected to provision a demand at the source and the destination of the demand in case the demand is satisfied ($L(dx) = 1$), and (5) provides routing continuity for each demand for each wavelength in the network. (6) ensures that no demand exceeds the hop count constraint on the lightpath, which is set to four hops. Finally,provides the lightpath availability constraint ensuring that only available lightpaths are used. The objective function is given as

$$Min : .(1 - L(dx)) + \beta \cdot . \quad (8)$$

The first term maximizes the total number of demands provisioned, while the second term minimizes the total capacity used. We use a scaling factor β which is a small value to ensure that the ILP first maximizes the total demands served, and then minimizes the capacity used by these demands

6 .1 Computation Module

Server started:

[DEBUG-NetworkModuleImpl]Entering:
NetworkModuleImpl(boolean isServer, ModuleManagement layerManagement)
[DEBUG-NetworkModuleImpl]Entering: start()
[DEBUG-NetworkModuleImpl]Entering: initSelectorParams()
[DEBUG-NetworkModuleImpl]| isServer: true
[DEBUG-NetworkModuleImpl]Entering: startSelectorThread()
[INFO-NetworkModuleImpl]| Listening for Events
[DEBUG-SessionModuleImpl]Entering:
SessionModuleImpl(ModuleManagement layerManagement)
[DEBUG-SessionModuleImpl]Entering: start()
[DEBUG-SessionModuleImpl]Entering: run()
[INFO-SessionModuleImpl]Waiting for new Messages
[INFO-TopologyInformation]NetworkSize: 15
[INFO-TopologyInformation]Starting thread to listen for topology updates on port 5189
[INFO-ThreadPool]Initializing Thread Pool, size = 5
[INFO-ThreadPool]Worker Thread 0 initialized
[INFO-Worker-Thread-0]Initializing Worker Thread ID = Thread-0
[INFO-ThreadPool]Worker Thread 1 initialized
[INFO-Worker-Thread-1]Initializing Worker Thread ID = Thread-1
[INFO-ThreadPool]Worker Thread 2 initialized
[INFO-Worker-Thread-2]Initializing Worker Thread ID = Thread-2
[INFO-ThreadPool]Worker Thread 3 initialized
[INFO-Worker-Thread-3]Initializing Worker Thread ID = Thread-3
[INFO-ThreadPool]Worker Thread 4 initialized

[DEBUG-ThreadPool]Thread Pool Initialized
[INFO-Worker-Thread-4]Initializing Worker Thread ID = Thread-4

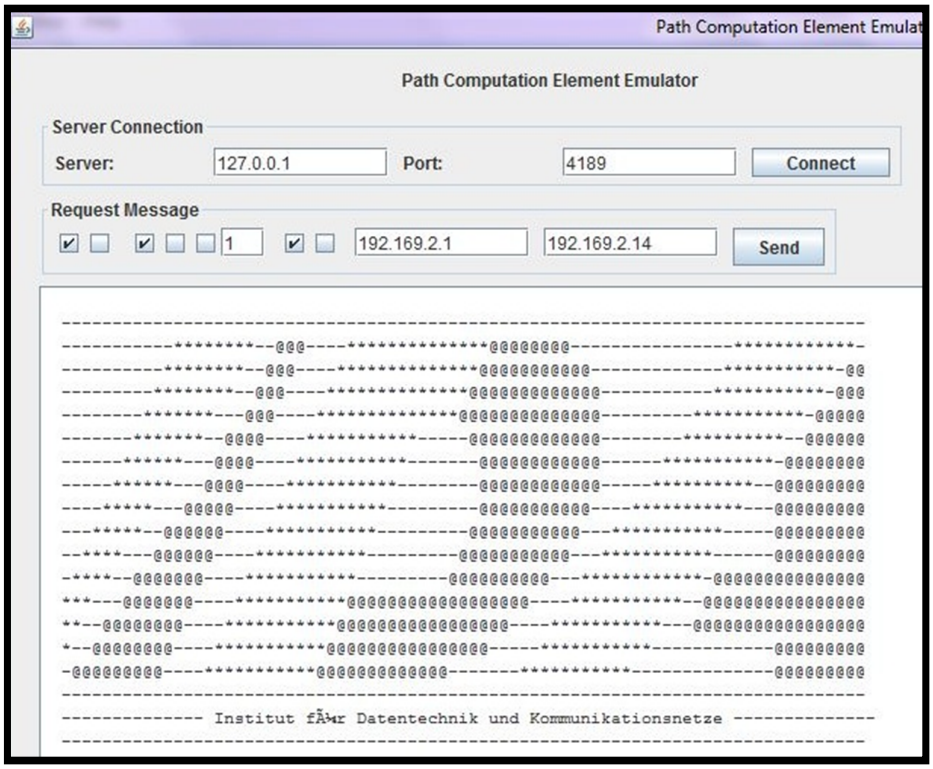


Figure (1 .6): This figure shows the connection to the server

Client started: 6.2

]DEBUG-NetworkModuleImpl]Entering: NetworkModuleImpl(boolean isServer, ModuleManagement layerManagement(
]DEBUG-NetworkModuleImpl]Entering: start()
]DEBUG-NetworkModuleImpl]Entering: initSelectorParams()
]DEBUG-NetworkModuleImpl]| isServer: false

]DEBUG-NetworkModuleImpl]Entering: startSelectorThread()
]INFO-NetworkModuleImpl]| Listening for Events
]DEBUG-SessionModuleImpl]Entering:
SessionModuleImpl(ModuleManagement layerManagement(
]DEBUG-SessionModuleImpl]Entering: start()
]DEBUG-SessionModuleImpl]Entering: run()
]INFO-SessionModuleImpl]Waiting for new Messages
]DEBUG-ClientModuleImpl|[
]INFO-ClientModuleImpl]Entering:
ClientModuleImpl(ModuleManagement layerManagement(
]DEBUG-ClientModuleImpl]| layerManagement:
com.pcee.architecture.ModuleManagement@5e9f23b4
]INFO-ClientModuleImpl]Entering: start()

The ruslt shortest path Path = 9 <- 7 <- 5 <- 3 <- 1

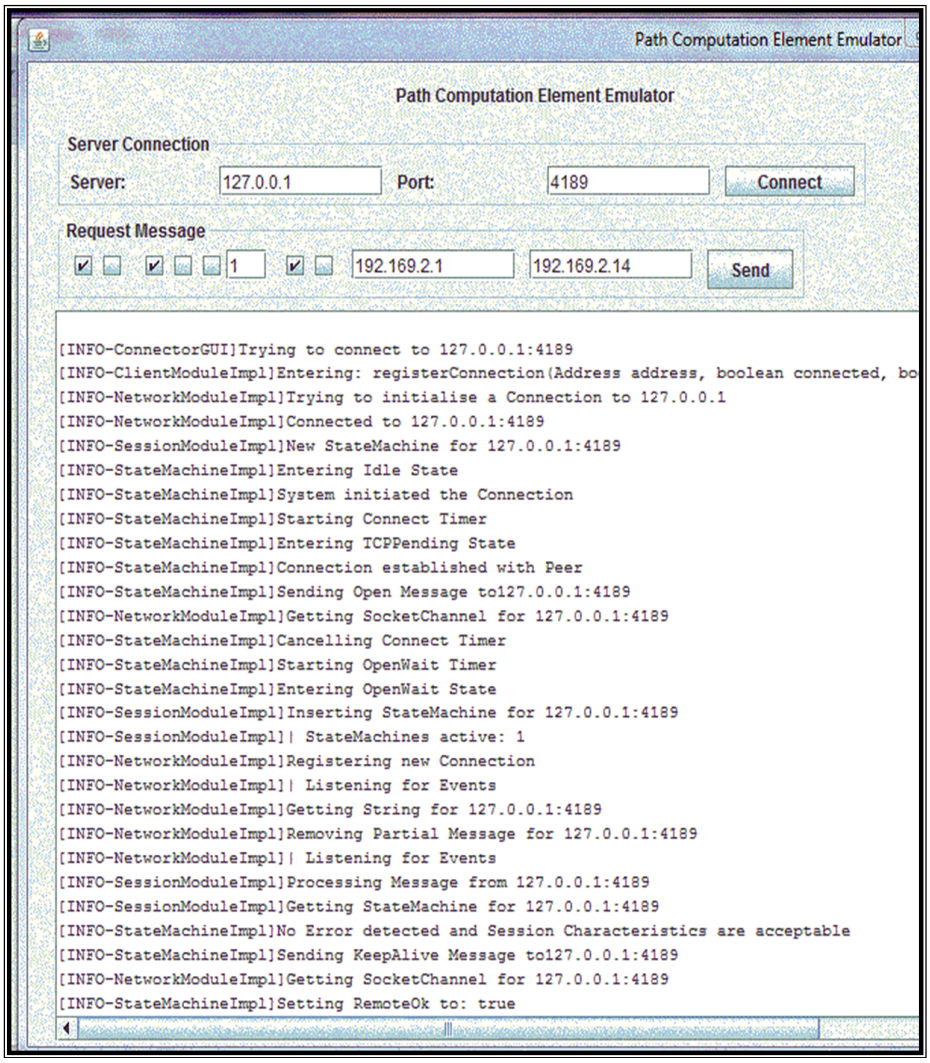


Figure (2 .6): This figure shows the connection to the server

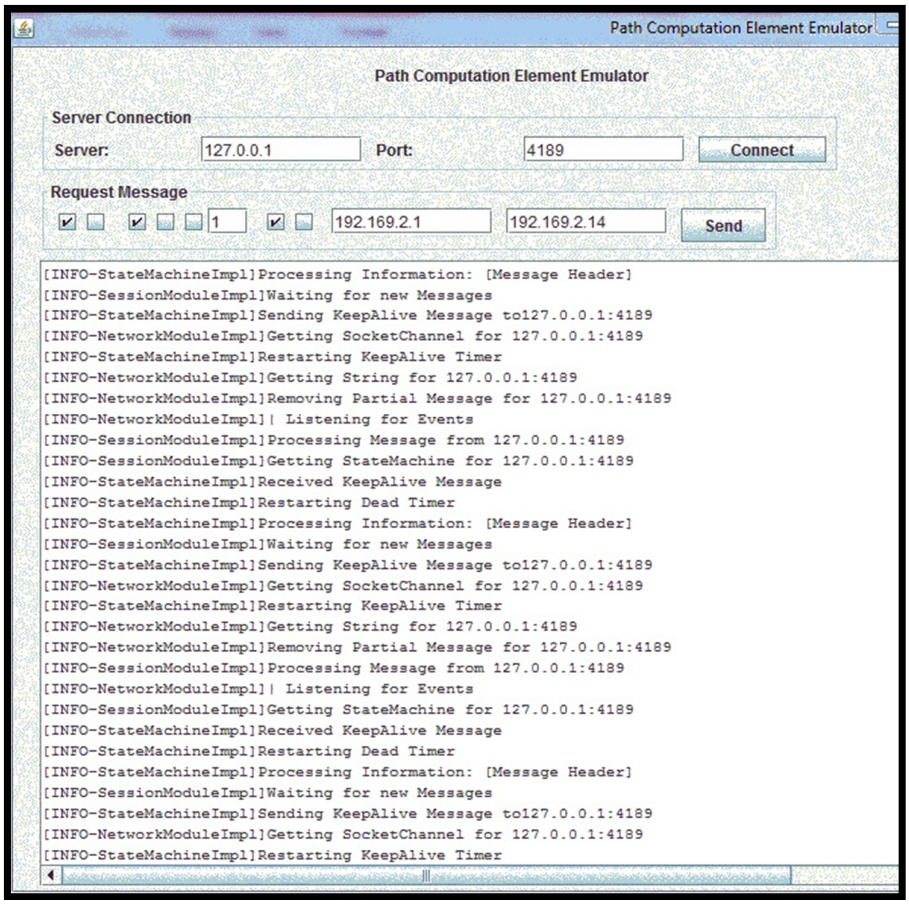
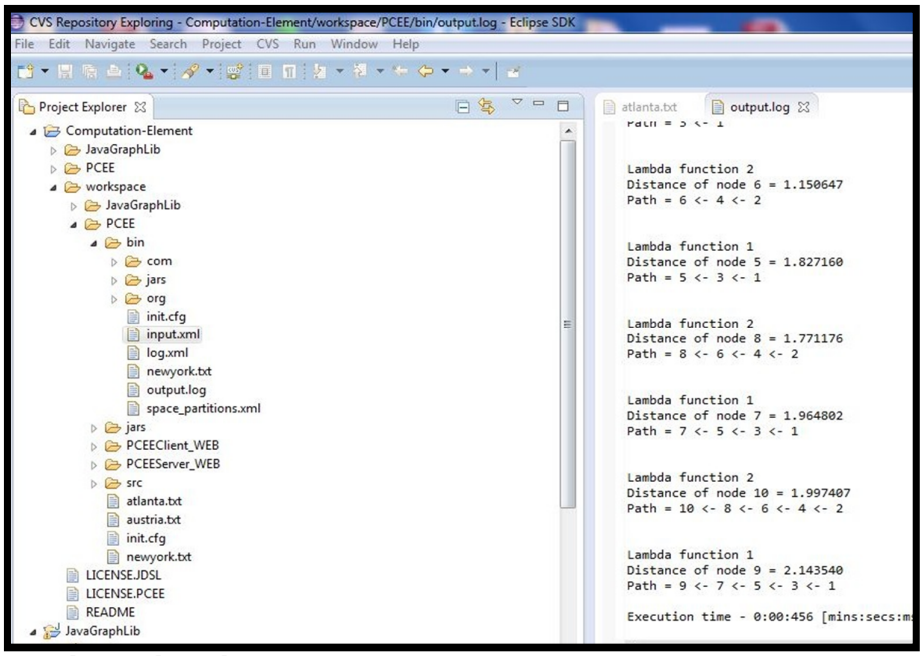


Figure (3 .6): This figure illustrates the transmission and reception process



Figure(4 .6): This figure Lambda function

PCE Application: ParallelPath Computation Dijkstra simulation Multi-Layer PCE

The ruslt shortest path Path = 9 <- 7 <- 5 <- 3 <- 1 [Execution time - 0:00:546 mins:secs:msecs]

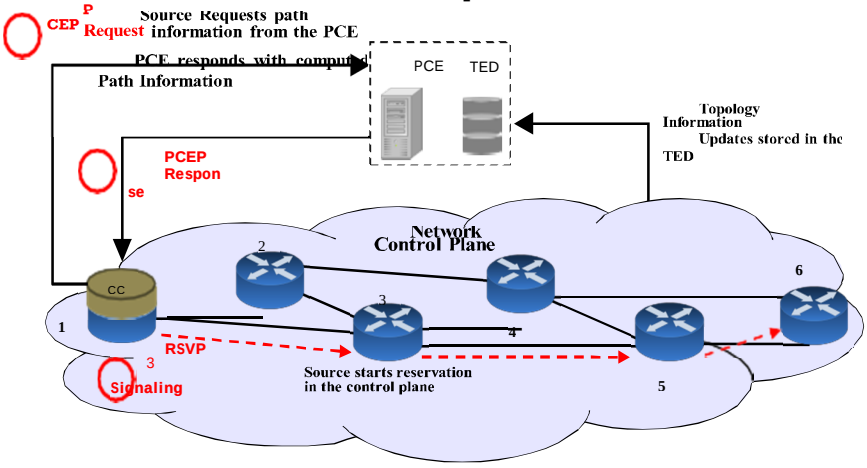


Fig. (1.5) Use of Path Computation Element (PCE) in transport networks

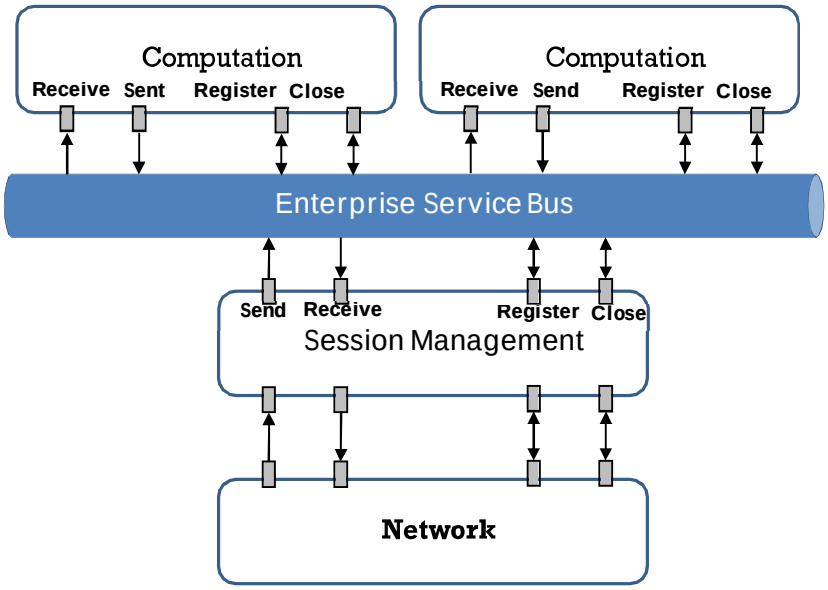


Fig (2.5) Support for multiple computation modules using a web-services based oad-balancing module

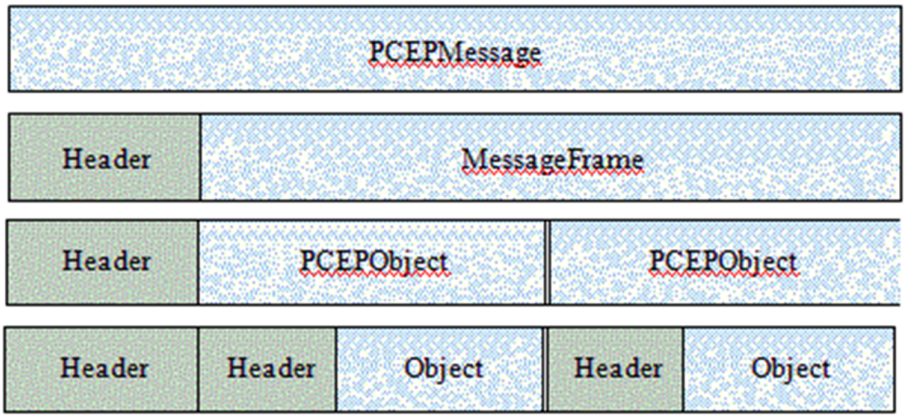


Fig (3.5) the pcep protocol represented as an object – oriented hierarchy

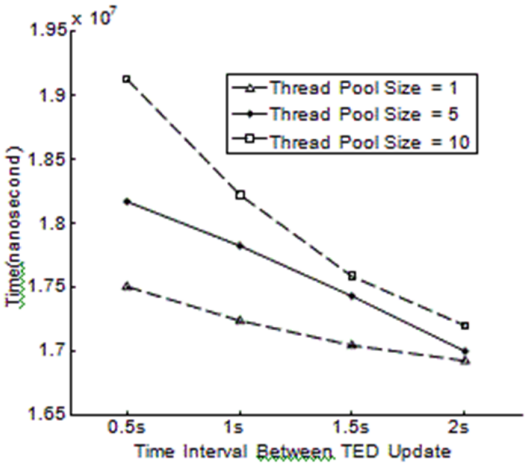


Fig. (4.5) Average total time taken by the computation module versus increasing time interval between TED updates under different thread pool size.

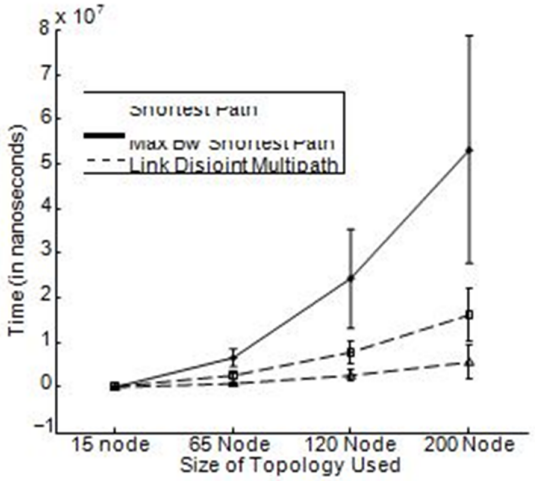


Fig. (5.5) Measurements for processing times in the computation module for different path computation algorithms and topology sizes.

7 .Conclusions:

In this paper, the pathological component of the PCE has been widely studied under the European Union project PACE. However, much work needs to be done when it comes to a multi-layer PCE structure. The calculation algorithm part of PCE has been widely discussed. A parallel algorithm has been proposed to solve this problem. PCE uses the Dijitsktra algorithm to find the shortest path of a single topological PCE structure. In a multi-layer PCE structure, you modify the Dijitsktra algorithm to accommodate the different layers of the topographic hierarchy. To create the parallel path calculation algorithm, use the lambda function in the python code for the hierarchical serial number. The lambda function relies on minimizing the map, which assigns different parallel tasks to find the shortest path of the hierarchy and reduce the result by choosing the shortest path from the various computed tasks. By providing a parallel calculation algorithm for multilevel PCE, the complexity of the computational problem has been solved and performance has increased.

References:

[1] Path Computation Element (PCE)—IETF RFC's 4665 and RFC 5

[2] IETF PCE Working Group, http://datatracker.ietf.org/wg/pce/charter/ Zhao, et al., "Extensions to the Path Computation Element Communication Protocol (PCEP) for Point-to-Multipoint Traffic Engineering Label Switched Paths", IETF RFC 6006, September 2010, http://tools. ietf.org/rfc/rfc6006.txt

[3] Farrel, J. P. Vasseur, J. Ash, "A Path Computation Element-Based Architecture", IETF RFC 4655, August 2006, http://tools.ietf.org/rfc/ rfc4655.txt

[4] http://www.cs.bu.edu/brite/

[5] Java Network I/O(NIO), http://download.oracle.com/javase/1.4.2/docs/ api/java/nio/package-summary.html

[6]E. W. Dijkstra, "A note on two problems in connexion with graphs," Numerische mathematik, 1959.

[7] Chamania, M.Drogon, A.Jukan, " Lessons Learned From Imple- menting a Path Computation Element (PCE) Emulator," (Postdeadline paper) Technical Digest of Optical Fiber Communication Conference 2011 (OSA/OFC 2011), Los Angeles, CA, March 2011

[8] https://en.wikipedia.org/wiki/Personal_consumption_expenditures_price_index

[9] https://fred.stlouisfed.org/series/PCE

[10] https://www.frbsf.org/.../pce-personal-consumption-expenditure-pric.