Development of an Interface between Signal Controller and Traffic Simulator

Ashutosh Bajpai
Research Assistant
Civil Engineering, IIT Bombay
Maharashtra, 400076 India
+918828828496
ashutosh.bajpai@iitb.ac.in

Tom V Mathew
Associate Professor
Civil Engineering, IIT Bombay
Maharashtra, 400076 India
+912225767349
vmtom@civil.iitb.ac.in

ABSTRACT: Adaptive Traffic Control algorithm is an important strategy to manage traffic at an intersection. These are an improvement of vehicle actuated signal control, where explicitly strategies are formulated to compute the signal timing considering the current traffic state obtained from sensors. However, field evaluation of these strategies is cumbersome and expensive and hence simulators which model traffic system can be a good alternative. The main challenge in this is a good interface between the signal control system and the traffic simulators. The signal control system needs the state of the junction in terms of vehicle occupancy at every instant. On the other hand, traffic simulator needs information on whether the signal state has changed. This two way communication requires an efficient interface which is similar to client-server architecture. The simulator acts as the server where as the adaptive control strategy act like client. This paper proposes an efficient interface to couple adaptive control strategy and traffic simulator. This interface mediates between traffic control system and traffic simulator and provides online interaction to simulation from the control strategy. This interface facilitates pure procedural routines to communicate and is written in C language along with Python/C API. Additionally, a module to estimate the vehicular delay due to the control strategy is developed. This delay is estimated by defining effective length of queue, which is provided as a user input.

This interface is tested using SUMO (Simulation for Urban Mobility), which is an open source, microscopic, space continuous and time discrete simulator developed by German Aerospace Centre. The traffic control strategy is analogous to the HCM vehicle actuated traffic control except that there is a queue prediction model which computes upper limits on the maximum green time. An isolated four arm junction having four phases is simulated for various flow conditions. The simulator supplied the state of the downstream detector to the traffic control algorithm at every simulation step and the control algorithm determines the signal time strategies (phase termination, green extension, and maximum green time). These strategies are communicated to the simulator. These communications were facilitated by the proposed interface. The average stopped delay is computed as the performance parameter. The interface was also coupled with another traffic simulator (VISSIM) and the results are compared. This interface justifies the concept of reusability by the evaluation of number of control strategy.

Key words: Traffic simulator, Signal controller, Procedural Interface, SUMO (**S**imulation of Urban Mobility), VISSIM (Verkehr In Städten - SIMulationsmodell).

1. Introduction

Increasing traffic congestion is a source of time loss and expense to users and managers of transportation systems. The transportation professionals and agencies are persistently searching for ways to mitigate urban traffic congestion, while minimizing costs and maintenance requirements. One of the effective ways of managing traffic is by using traffic signals which separates conflicting movements and assigns time for these movements. However, in urban areas, traffic signals limit the road capacity resulting in congestion. Mitigating traffic congestion, therefore, relies heavily on an efficient and well-managed traffic signal control system.

Most signal control systems determine signal timings for various time-of-day such as morning and evening peak. Signal timing plans, however, are inflexible and do not change during each of these time-of-day intervals and are hence called *fixed-time* plans. These

operate with the fundamental assumption that volumes are stable and one cycle length remains optimal throughout a design period. However, such volume stability is rare and traffic usually builds to a peak and then gradually subsides during a period. This implies that fixed time controller may operate optimally for a short interval when current volumes match the design volumes and sub-optimally during other intervals.

To address the above limitation, vehicle actuated traffic controllers are being widely used which respond to current traffic state obtained from vehicle sensors and various studies have reported significant improvement in the performance [1, 6]. The main feature of all most all actuated controllers is the ability to adjust the duration of the active phase in response to the traffic flow. Green time for a phase is a function of the traffic flow and can be varied between pre-specified minimum and maximum green times. Although they operate in real time, the signal timing cannot be claimed as optimal. Adaptive traffic controllers are, therefore, proposed to improve the performance of the vehicle actuated controller by computing the timing based on the predicted traffic state. The traffic state can be predicated from the upstream vehicle sensors for the current cycle or stop-line detection for next cycle.

Several algorithms proposed for adaptive traffic controller's claims superiority. Evaluation of these control algorithms by implementing in the field is not feasible primarily because it affect the existing users, sometimes adversely. However, simulators which model the traffic system are a good alternative to field evaluation. The main challenge is to have a good interface between the signal control system and the traffic simulators. The signal control system needs the state of the junction in terms of vehicle occupancy at every instant. On the other hand, the traffic simulator needs state of the signal, also at every instant. This two way communication requires an efficient interface similar to client-server architecture. The traffic simulator acts as the server where as the signal controller acts like a client.

There is several engineering software that require similar interfaces [3-4]. For instance, asynchronous interface between ABAQUS and HEREZH++ is a good example of inter-process communication [3]. This paper proposes an efficient interface to couple traffic simulators and a signal controller. The remainder of the paper is structured as follows. First, a brief overview of the signal control algorithm and the traffic simulators used in this study are described. Then, the development and implementation aspects of the proposed interface are elaborated. Finally, evaluation of the control strategy using the developed interface is presented.

2. Signal Control

The signal control algorithm tries to reduce the average stopped delay of the vehicles queued up due to the signal. It defines phase terminating conditions using minimum and maximum green times and a threshold headway (gap between the vehicles). These minimum and maximum bounds for green time for each phase are adjusted based on the state of the phase in the immediate previous cycle and history of traffic data obtained from the past few cycles. The algorithm monitors the performance at the end of each phase and, if required, re-sets the green time bounds. In this strategy, prediction of queues is done by a linear regression model which is used to set the green time bounds. This control strategy enables the system to adapt to the stochastic and dynamic behaviour of traffic flow at the intersection [5]. Another important motivation for the development of this algorithm is to address the challenges posed by heterogeneous traffic. Heterogeneous traffic, observed in various cities of the developing countries, is characterized by non-lane based movement and mixed vehicle type [2]. Various attempts have been made to incorporate heterogeneous aspects [6]. The key features of this

system are downstream (stop-line) detection, variable loop size, logical grouping of phases, and dummy phases [6].

3. Traffic Simulator

Traffic models are mainly based on analytical or simulation approaches. The analytical models often use queuing theory, optimization theory or differential equations that can be solved analytically to model the road traffic. Although these models are mathematically sound, they often do not account variation of traffic system over time especially if the geometry is complex. On the other hand, traffic simulation models are able to respond to changes over time and use stochastic functions to reproduce the dynamics of a traffic system. Traffic simulation has become a powerful and cost-efficient tool for investigating traffic systems [7]. It can, for instance, be used for evaluation of different traffic management strategies. Traffic simulators offer the possibility to experiment in a safe and non-disturbing way with an existing or non-existing traffic system. Traffic simulation models are typically classified according to the level of detail at which they represent the traffic stream. Three categories are generally used, namely: *Macroscopic, Mesoscopic and Microscopic.* Macroscopic models use a low level of detail, both regarding the representation of the traffic stream and interactions. *Mesoscopic* models often represent the traffic stream at a rather high level of detail, either by individual vehicles or packets of vehicles. Microscopic models represent the traffic stream at a very high level of detail. They model individual vehicles and the interaction between them. Micro models consist of several sub-models called as *behavioural models* that handle specific interactions. The most essential behavioural model is the car-following model, which handles the longitudinal interaction between two preceding vehicles. Other important behavioural models include models for lane-changing, gap acceptance, overtaking, ramp merging, and speed adaptation. The intersection modelling in a simulator refers to the control and flow of each vehicle at an intersection. For intersection modelling, a traffic simulator has two major blocks - a traffic simulation model and a signal controller module. The simulator will represent the real world field condition of how the vehicles move in a road network. The virtual detector placed in the simulated network will give the vehicle arrival data to the signal control module. The signal control module in turn receives all the data and computes the signal timings at specified time. These times are given back to the traffic simulator to adjust its signal timing. Major building blocks of the simulator include: vehicle representation, vehicle generation, mid-block model (vehicle following model, lane changing, and vehicle movement), intersection model (signal system, turning movement, and vehicle movement), detector simulator, and data extractors [5].

4. Interface between Signal Controller and Traffic simulator

A traffic simulator imitates the real behaviour and characteristics of the traffic pattern with the amalgamation of different traffic models like car following model and lane changing model. An interface allows a user to set traffic parameters for a specific control strategy as well as retrieve the state of the signal.

Most of the issues pertaining to interface development such as design, security issues, modularity, and easy-to-devise can be addressed by following client-server architecture. Several software development life cycle issues need to be considered while designing the interface which includes: reusability, modularity, abstraction, and fault tolerance. Proposed interface cover most of these issues.

4.1 Interface architecture

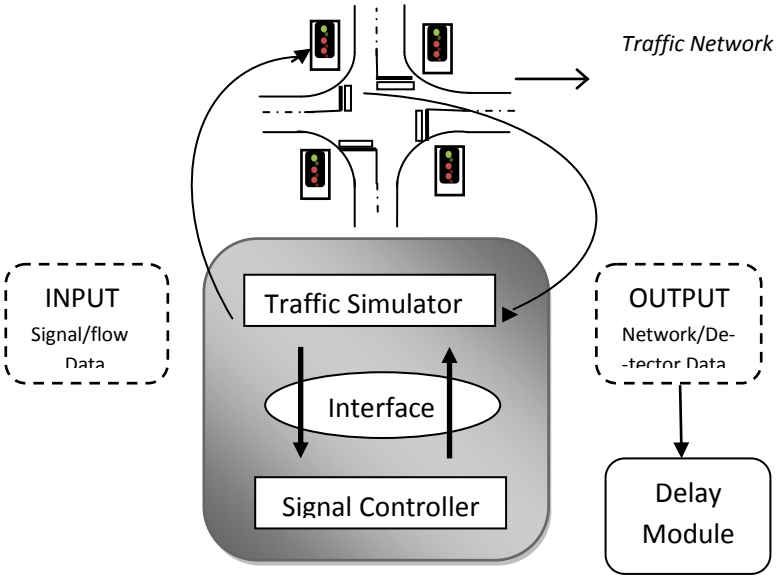

Fig.1 Architecture of the interface which couples signal controller and traffic simulator

The interface provides communication between the two independent processes by granting access to *set* and *get* states. In the current problem, traffic simulator grants access to the signal controller to set the signal states and get the vehicle detector state. In this two-way communication a user defined control strategy can pass and retrieve the information using the procedural members of the interface.

Signal control algorithms needs flow, occupancy, and detector state to optimally allocate green time. The retrieving procedures should facilitate extraction of such information from traffic simulator in terms of detector output or signal state. On the other hand, signal control algorithm sends some packets carrying information regarding phase timing or termination condition and the interface should have procedures to set them. Stopped delay is the output which is generated by interface by calling some procedures. The architecture of the interface which couples signal controller and traffic simulator is shown in Fig 1.

4.2 Solutions for the interface communication

The interface which couples the signal controller and the traffic simulator should exchange data rapidly. Inter process communication (IPC) provides a mechanism for exchanging data between processes (either on the same or different networked computers) and enables communication between applications even though they may be written in different languages and for different operating systems. The problem associated with inter process communication is handled by an efficient transaction control. This transaction control is achieved by developing the system in half-duplex mode which allows only one way communication in a given instant. In other words, if the signal control algorithm retrieves data using the interface procedure, then it cannot simultaneously set any other traffic parameters. The half-duplex mode also provides an efficient mechanism for collision control. The solution to the communication problem between two different grammars is achieved by python/C API. Python is a programming language that allows quick implementation and easy integration. It also allows dual development approach; top down (procedural) and bottom up (object oriented) approach.

4.3 Estimation of Intersection delay

Vehicular delay is one of the important parameter to evaluate the performance of the signalized intersection. Highway Capacity Manual (TRB, 2000) is using delay as the performance indicator to divide the level of service of a signalized intersection and the same is used in estimating the optimum cycle length. The delay models proposed by HCM are empirical in nature and gives aggregate delay values. However, accurate delay values can be estimated from the simulator [6]. Such a delay measure will be useful in modelling and evaluation of intersection models and traffic controls systems. Although the concept of delay is simple, several issues arise while implementation. For example, apart from traffic signal, delay may occur due to various factors including quality of progression, traffic volumes, and intersection capacity. Here, we qualify the delay as stopped delay and defines the average duration a vehicle stops at an intersection having traffic signal. Two issues arise here. First, the effective length that needs to be considered upstream of the intersection for each approach here it is user defined with default value of half of the lane length. It assumes that the signal has effect only in this section. However, if this length is higher than the lane length, the later is used. Second, often vehicles crawls near the intersection and considering absolute stopping time may be unrealistic. Hence, the stop actually means vehicle moves at a crawling speed. The user has to specify this speed and vehicle in the 'effective length' have a speed lower than the crawling speed would be considered as waiting. This algorithm is divided into three procedures, namely EFFECTIVE LENGTH, DELAY_INTERSECTION, and GET DELAY.

4.3.1 Procedure: EFFECTIVE LENGTH

The first procedure is to compute the effective length of each lane considering user input. Default value of effective length is half of the lane length in case of not defined. The variables used in this procedure are given below.

1. *user_Def_Len* – variable containing value of lane length defined by user.
2. *effective length* – temporary variable containing value of effective length.
3. *lanes_Length* – containing value of length of lane.

The algorithm involves the calculation of effective length of a lane which is used in calculation of intersection delay. It gets called from DELAY_INTERSECTION procedure. Pseudo code for this procedure is given in Fig 2.

```
PROCEDURE:    EFFECTIVE LENGTH (lanes_Length)
1: if Define user_Def_Len
     11: Initialize variable user_Def_Len
     12: if   user_Def_Len >=: lanes_Length
                    effective length =: lanes_Length
              else effective length =: user_Def_Len
2: if NotDefine user_Def_Len
     effective length =: lanes_Length /2
3: return effective length
```

Fig.2 Effective length algorithm in intersection delay calculation

4.3.2. Procedure: DELAY_INTERSECTION

Second procedure estimates the intersection delay for each junction defined by user. Threshold speed is also the input to the procedure from user side else default value is 0.5 m/s^2. This procedure also integrates the other two procedures. The variables used in this procedure are given below.

1. *junction_Id* – variable containing the id's of all junctions in network.
2. *user_Junction_Id* – variable containing the ids of all junctions defined by user.
3. *lanes_Id* – used to contain the ids of lanes related to specific junction.

4. *lanes_Length* – accommodating the length of each lane.
5. *eff_Len_Lanes_Id* – hosting value of effective length for each lane
6. *veh_Id* – variable contains the ids of all vehicles in a selected lane.
7. *pos_Veh_Id* – variable hosts the position of each vehicle in a lane.
8. *speed_Veh_Id* – getting the speed of all vehicle which cleared position criteria.
9. *counter_Lanes_Id* – contain the total waiting step for a lane.
10. *nof_Veh_Lanes_Id* – hosts the total no of vehicle in each lane.
11. *th_Speed – user defined.*

This procedure is main link to call the algorithm to get intersection delay which communicates with other two procedures. First it calls the EFFECTIVE_LENGTH procedure then calculates the delay on each lane individually and then transfers the call to GET_DELAY procedure to get intersection delay. It also generates the network level delay. Pseudo code for this procedure is given in Fig 3.

```
PROCEDURE:    DELAY_INTERSECTION (simulation)
1:   Initialize variable junction_Id
2:   Initialize variable user_Junction_Id and th_Speed
2:   for j ← 1 to Len (junction_Id)
        21:   for k ← 1 to Len (user_Junction_Id)
                211:   if (junction_Id[j] = =: user_Junction_Id[k])
                            Initialize lanes_Id[k]
                212:   for i← 1 to Len (lanes_Id[k])
                            Initialize lanes_Length[k][i]
3:   for each Cycle
        31: for each lanes_Id
                eff_Len_Lanes_Id =: effectiveLength (lanes_Length)
4:   for each simulation step
        41: for k ← 1 to Len (user_Junction_Id)
                411: for j ← 1 to Len (lanes_Id[k])
                        Initialize veh_Id[k][j]
                        4111: for i ← 1 to Len (veh_Id[k][j])
                                Initialize pos_Veh_Id[k][j]
                                41111: If pos_Veh_Id[k][j] >=: eff_Len_Lanes_Id[k][j]
                                        Initialize speed_Veh_Id
                                        Step 411111: If speed_Veh_Id < th_Speed
                                                counter_Lanes_Id[k][j] ++
5:   Initialize nof_Veh_Lanes_Id for each lane
6:   getDelay (nof_Veh_Lanes_Id, counter_Lanes_Id)
```

Fig.3 Delay intersection algorithm in intersection delay calculation

4.3.3 Procedure: GET DELAY

Third and last part of the algorithm is responsible to generate the weighted some of the lanes delay and find out the intersection delay. The variables used in this procedure are given below.

```
PROCEDURE:    GET DELAY ( nof_Veh_Lanes_Id,  counter_Lanes_Id)
1: for k ← 1 to Len (user_Junction_Id)
      11: for j ← 1 to Len (lanes_Id[k])
            d_Lanes_Id [k][j] =: counter_Lanes_Id [k][j] / nof_Veh_Lanes_Id[k][j]
   12: for j ← 1 to Len (lanes_Id[k])
         td_Junction_Id[k] =: td_Junction_Id + d_Lanes_Id [k][j] * nof_Veh_Lanes_Id[k][j]
       tnof_Veh_Lanes_Id[k] =: tnof_Veh_Lanes_Id + nof_Veh_Lanes_Id[k][j]
   13:  d_Junction_Id[k] =: td_Junction_Id[k] / tnof_Veh_Lanes_Id[k]
2: return d_Junction_Id
```

Fig.4 Get delay algorithm in intersection delay calculation

1. *d_Lanes_Id* – having value of delay for each lane.
2. *td_Junction_Id* – temporary variable using for weighted sum.
3. *tnof_Veh_Lanes_Id* – temporary variable containing value of total no of vehicle passed from a junction.

4. *d_Junction_Id* – having value of delay for each junction.

This procedure gets called from the DELAY_INTERSECTION and computes the weighted sum of the lanes delay in order to compute intersection delay. Pseudo code for this procedure is given in Fig 4.

5. Implementation of the Interface

Proposed interface is implemented for SUMO traffic simulators. SUMO is an open source traffic simulator [9] and Python/C API is used to develop the interface. The implementation framework is shown in Fig 5. Python/C API provides an efficient way to integrate two different grammars [10]. The API is used to design the interface because it can handle both procedural and non-procedural client.

Other implementation issues are: software optimization, fault tolerance, and credibility. Optimization is achieved by modular design at the macro as well as unit level. Exception handling is done for each procedure of the interface so that at the event of any instruction failure user will get complete information on error. Although the system does not have automated fault tolerance capability, static fault tolerance is well achieved by exception handling. Credibility of the system is ensured by the use of only open source Python/C API and not using any third party software. Whole implementation is divided into modules which facilitate further amendments.

This interface is one of the good examples of abstraction where the only essential information is passed to the end user. Most of the end users of the interface are traffic professionals and hence interface is abstracted. This interface provides the abstracted members or procedures to send or retrieve data from client application. On the other hand, the implementation also ensures reusability and can be used by both client language C and C++ [11] and can be used with java with some further modification. The approach of using procedural members has been an efficient technique in software development over the years.

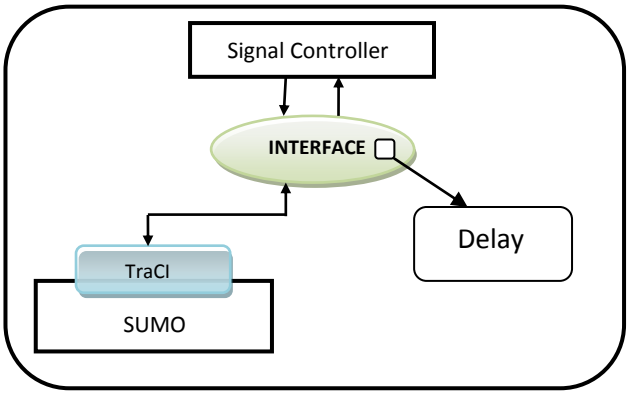

Fig5. Interface Architecture for coupling Signal Controller and SUMO

The architecture of the proposed interface is independent of the traffic simulator. Signal control code file is the root file which transfers the control to the interface file depending on which simulator is used for the evaluation. There is separate interface file to handle all the control for traffic simulator. File hierarchy is very easy to use and designed in order to be simpler for the end user. For instance, SC.CPP is the main file containing signal control algorithm. SUMOinterface.h is the implementation file for SUMO traffic simulator which uses the Python/C API. TraCI is the default interface in python provided by SUMO (Fig 5).

Stopped delay algorithm is written in separate module using python in sense to achieve modularity and by some procedures of interface access is allowed.

5.1 Implementation of interface on SUMO traffic simulator

In order to implement the interface for coupling adaptive traffic control strategy and traffic simulator an open source, efficient traffic simulator SUMO [12] is used as the one platform. TraCI [13] is the short form for "Traffic Control Interface" in SUMO. The basic idea behind is to give access to a running traffic simulation. TraCI uses a TCP based client/server architecture to provide access to SUMO. Thereby, SUMO acts as server that is started with additional command-line options. After starting SUMO, a client connects to SUMO by setting up a TCP connection to the appointed SUMO port. The client application sends commands to SUMO to control the simulation run, to influence single vehicle's behaviour or to ask for environmental details. SUMO answers with a status-response to each command and additional results depend on the given command. A TCP message acts as container for a list of commands or results. Each TCP message consists of a small header that gives the overall message size and a set of commands that are put behind it. The length and identifier of each command is placed in front of the command. TraCI provides the front-end designed on python commands. In the proposed interface, responses from server are python based, but client code for this implementation of interface can be object oriented (C++) or procedural (C) (Fig. 3). This communication between two different grammars is achieved by inter process communication designed by Python/C API. This interface is also a good example of application of python/C API, provided by python community and it is open source.

File hierarchy

While coupling the adaptive control strategy and traffic simulator, one of the important issues is management of files. SUMO contains number of files to run the traffic simulation. Some of the important files required for GUI interface and TraCI are discussed here. Later some description of interface files and how to use those files is presented. The complete file hierarchy is divided in to three modules (Fig. 6). First, to run the traffic simulator SUMO the four basic steps related to file structure are Build your network, Build the demand, Dynamic user assignment and perform the simulation.

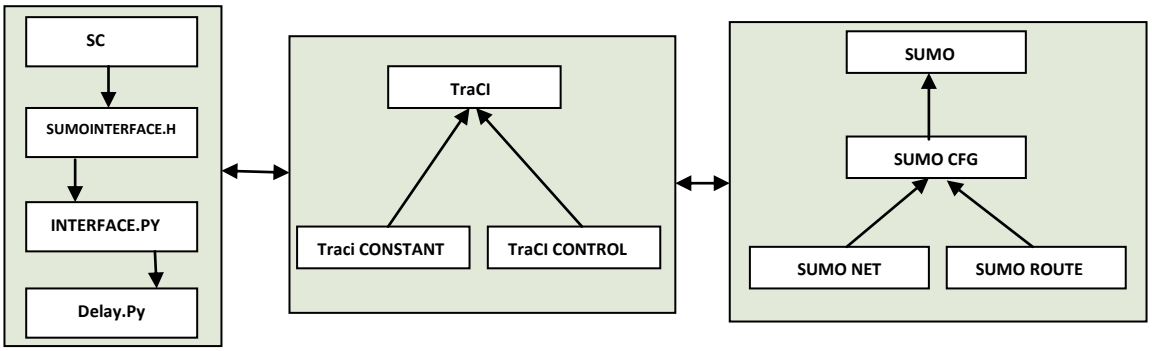

Fig6. File hierarchy for coupling Signal Controller and SUMO

Second hierarchy defines the file structure of TraCI for making online interaction with simulation. Two important files associated with it are TraCI control and TraCI constants. These are front end files for TraCI software.

The third and last hierarchy defines the file structure of interface. SC.CPP is the main file containing the core signal control. From this file, a user can call the various procedures of

interface to link with traffic simulator. The core files associated with interface implementation are header file SUMOINTERFACE.H, python script Delay.py and python file INTERFACE.py. The later is used to define SUMO related traffic parameters setting and facilitates user to modify according to their requirements. Delay.py is the implementation of Stopped delay algorithm which is written in a separate module. SUMOINTERFACE.H is the main file which implements the python/C interfacing using python/C API. This file can be directly associated with any control file written in C or C++. Transferring the control from one file to other file in the given file hierarchy by invoking a single procedure is shown in the pseudo code below.

```
   //pseudo Code for single procedure
1. SUMO("init",0)
2. If Def SUMOinterface.h else goto 6
3. If Def ATCS.py else goto 5
4. def initialize_SUMO()
5. ATCS.py not found goto 7
6. SUMOinterface.h not found
7. exit
```

5.2 Members of Interface

An interface is the pool of carriers which proceeds to send or retrieve data from one process to the other. In the current development, there are a number of procedures implemented to *set* or *get* traffic parameters value and the most important aspect of the interface is the member's prototype is user friendly. Here we are discussing only few members and their prototype.

Procedure Simulation_Init:

This member of the interface is a must as it initiates the SUMO object of GUI. If SUMO is already in OS environment, then it links the path to that SUMO object. Otherwise it creates a new SUMO object. Description of the path of SUMO configuration related files can be modified in INTERFACE.py file.

```
//Procedure Prototype
//Procedure Prototype
SUMO("init",0);
```

Procedure Single_Step:

The member Single_Step executes the SUMO for a single step. In SUMO the minimum step size (state updating interval) is 1 second. So this procedure must be called for a number of times when user wants to run SUMO.

```
//Procedure Prototype
SUMO("run_step1",0);
```

Procedure Assign_Green:

This member of the interface is responsible to *set* data on traffic lights. It assigns the green signal to the phase plan define by the user in INTERFACE.py for SUMO.

```
//Procedure Prototype
SUMO("a_green",Variable);
```

6. Evaluation of a Control Strategy by the Developed Interface

The quality and the reliability of the proposed interface are assessed by an evaluation of a signal controller algorithm by several test cases. In order to investigate the effect and validation of information transfer on the simulation run times, these tests go through a comparison based testing. This control strategy is evaluated on two different simulation models. The first one is performed on SUMO traffic simulator by the proposed interface and compared with the other test case performed on other traffic simulator VISSIM [8]. The evaluation of control strategy is based on the intersection delay generated by the traffic simulator, an average waiting time of a vehicle at an intersection due to signal control. Steps adopted to test this interface are listed below:

(a) Traffic composition and characteristics

In this evaluation traffic composition and characteristics is used from a previous study [14]. Composition of mixed vehicle and the characteristics of different vehicles are shown in the Table 1:

Table 1 Traffic Characteristics and composition

Vehicle type	Length (m)	Width (m)	Desired Speed (km/h)	Acceleration (m/s^2)	Deceleration (m/s^2)	Composition (%)
Truck	10.21	2.50	20	0.80	0.60	1.60
Bus	11.54	2.50	20	0.80	0.60	3.80
LCV	06.10	2.20	25	0.70	0.60	0.90
Car/jeep	04.40	1.50	30	1.20	1.00	20.6
Three-wheeler	03.20	1.40	25	0.90	0.80	22.0
Two-wheeler	01.80	0.50	40	1.70	1.20	43.0
Rickshaw	02.70	0.95	15	0.20	0.20	5.00
Bicycle	01.90	0.45	05	0.30	0.30	3.00

(b) Network preparation for traffic simulator

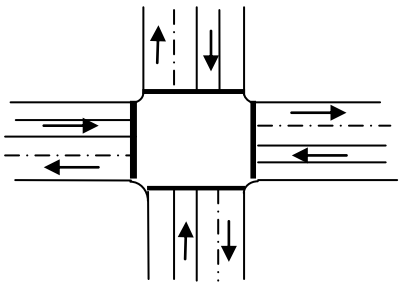

Fig.7 Isolated junction

Table 2 Network Characteristics

No of intersection	1
No of incoming edges	4
No of outgoing edges	4
No of lanes in incoming edge	3
No of lanes in outgoing edge	1
Length of lane	500 m
Maximum speed on lane	13.9 m/sec

For the evaluation of the control strategy for a traffic junction requires a network file, which is the input for the traffic simulator. The syntax of the network file is specific to the simulator and usually tools provided by these simulators can convert external formats to their native ones. The network configuration is as shown in the Table 2. Geometry of network is shown in Fig 7.

(c) Performing the simulation

Simulation is performed for 3600 seconds with different vehicular flows. At the start of simulation, an instance of SUMO/VISSIM is invoked by the control strategy through the procedural member "simulation_init" of the interface. Then, based on certain conditions specified by the control strategy the simulation is performed. Other members of the interface will be called from signal controller to execute the control strategy in simulator.

(d) Comparison of delay produced by SUMO and VISSIM

The traffic simulator is calibrated before using which ensure the simulator give the field values. The performance parameter used for the evaluation is the average stopped delay per cycle per vehicle. Delay values obtained from the SUMO is comparable to delay obtained from VISSIM. The comparison results are tabulated in Table 3 and are also shown in Fig 8. Both simulators gave comparable results for all the flow conditions tested. However, lower flow values gave relatively higher error.

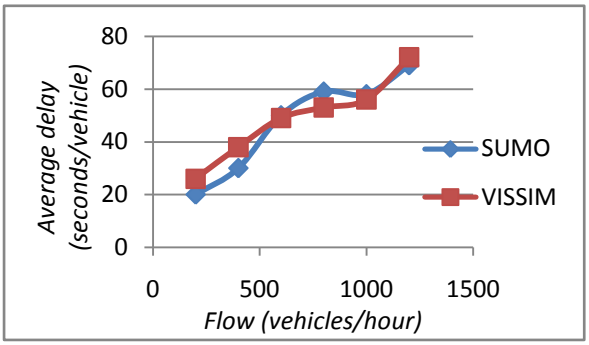

Table 3 Comparison of Delay

Flow (vehicle/hr)	Average delay (sec/vehicle)	
	SUMO	VISSIM
200	20	26
400	30	38
600	50	49
800	59	53
1000	58	56
1200	69	72

Fig 8: Comparison of delay obtained by SUMO and VISSIM

7. Conclusion

This paper proposes an efficient interface to couple traffic simulator and signal controller which facilitates real-time communication through pure procedural routines written in C and Python/C API. This interface is tested using SUMO (an open source simulator) and compared with other traffic simulator VISSIM (a proprietary simulator). The proposed interface enables signal control algorithm developers to evaluate their strategies by SUMO traffic simulator easily and effectively. The prototypes of different members for a traffic simulator are same which shows a efficient user friendly environment. The traffic control algorithm is analogous to a simple vehicle actuated traffic control except for a queue prediction model to compute maximum green time for each signal phase. To evaluate the proposed interface, an isolated four arm junction having four phases is tested for various flow conditions. Average stopped delay per vehicle is used as the parameter to evaluate the signal control algorithm. Testing results for various traffic flow conditions yielded analogous delay values indicating robustness of the interface. Evaluation of signal control strategy by two different traffic simulators also demonstrates reliability, modularity and accuracy of interface. The proposed interface also established the feasibility of integration of various traffic simulators on a common platform in future. The proposed interface needs to be generalized by testing several other commercially available simulators to provide a common platform. Further, the library provides by this interface can be enhanced by adding modules for new members. Finally, this interface can be evolved as a standard protocol for traffic simulators and signal controllers.

Reference

1. J. N. Darroch, G.F. Newell and R.W.J. Morris. Queues for a Vehicle-actuated Traffic Light. Operations Research, 1964, 12(6) 882-895.

2. Muralidharan, V. Ravikumar, P. Bonala, S. Tagore, M. R. A composite signal control strategy for Indian roads, Indian Highways, 2006, 34(8), 21-34.

3. Rio, G., Laurent, H. & Bles, G. Asynchronous interface between a finite element commercial software ABAQUS and an academic research code HEREZH. Advances in Engineering Software, 2008, 39, 1010-1022.

4. G. Linden and A. I. Verkamo. 1995. An interface between different software development environments. In Proceedings of The 10th Knowledge-Based Software Engineering Conference (KBSE '95). IEEE Computer Society, Washington, DC, USA, 79-..

5. Nalge, M. Nair, R. Kancharla, C. Mathew, T. V. Rangaraj, N. Real-Time Adaptive Control Policy for Intersection with Downstream Detection, 18th World congress on Intelligent Transport Systems, Orlando, USA. 2011.

6. Ravikumar, P., Mathew. T. V, Vehicle Actuated Signal Controller for Heterogeneous Traffic Having Limited Lane Discipline, ITE Journal, 81(5), 44-53, 2011

7. Shiraishi, T., Hanabusa, H., Kuwahara, M., Chung, E., Tanaka, S., Ueno, H., Ohba Y., Furukawa M., Honda, K., Maruoak, K., Yamamoto T. . Development of a Microscopic Traffic Simulation Model for Interactive Traffic Environment, In Proceedings 12th ITS World Congress. 2004

8. VISSIM User manual. User Manual, PTV AG 2008. http://www.ptvamerica.com/support /library.

9. Krajzewicz, D., Hertkorn, G., Rössel, C. and Wagner, P.. Sumo (simulation of urban mobility); an open-source traffic simulation. In Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM2002), 2002, pages 183–187.

10. Python/C API Reference Manual. Python v2.7.2 documentation. http://docs.python.org /c-api.

11. Strousstrup, B. The C++ Programming Language. Mass: Addison-wisely; 1986.

12. Krajzewicz, Daniel, Elmar Brockfeld, Jürgen Mikat, Julia Ringel, Christian Rössel, Wolfram Tuchscheerer, Peter Wagner and Richard Wösler. Simulation of modern Traffic Lights Control Systems using the open source Traffic Simulation SUMO. Proc. 3rd Industrial Simulation Conf. 2005; Berlin, Germany.

13. Wegener, A. et al. TraCI: An Interface for Coupling Road Traffic and Network Simulators. In Proc. of the 11th Communications and Networking Simulation Symposium 2008, CNS'08.

14. Mathew, T. V., Radhakrishnan, P. Calibration of Microsimulation Models for Non-lane-Based Heterogeneous Traffic at Signalized Intersections, Journal of Urban Planning and Development, ASCE, 2010, 136, 59

Biographies

Ashutosh Bajpai: Mr. Ashutosh Bajpai is working as a Research Assistant at Dean R&D, IIT Bombay. He has achieved his bachelor's in computer science and engineering with honours from Rajeev Gandhi Technical University, MP (India) by 2010. He is currently involved in ITS project entitled 'Second Generation Area Traffic Control System (CoSiCoSt-2G)' funded by DIT, under the supervision of Prof. Tom V Mathew. He is also one of the open source contributors for German based traffic simulator named SUMO. His research interests include traffic simulation, adaptive signal controllers, quantum computing and algorithm designing.

Tom V Mathew: Dr. Tom V. Mathew is working as an Associate Professor of the Department of Civil engineering, IIT Mumbai (India), and specialized in Transpiration Systems Engineering. Dr. Mathew completed the master of technology in transportation engineering from Indian Institute of Technology Madras, Chennai in 1993. In 1999 he finished the doctorate in Civil Engineering, from the same institute. His doctoral research was on 'bus transit route network design using genetic algorithm'. His research interests include transportation network design, traffic flow modelling, evaluation of the impacts of transportation systems, and applications of genetic algorithm, artificial neural networks, and cellular automaton in transportation.