User Manual for the Time-variant Community Mobility Model

Release Version: TVCM-1.1-beta

Release Date: May 31, 2007

Authors:
Wei-jen Hsu, Dept. of Computer and Information Science and Engineering, U. of Florida.
Thrasyvoulos Spyropoulos, INRIA, Sophia-Antipolis, France.
Konstantinos Psounis, Dept. of Electrical Engineering, U. of Southern California.
Ahmed Helmy, Dept. of Computer and Information Science and Engineering, U. of Florida.

Webpage:
http://nile.cise.ufl.edu/TVC_model/
http://nile.cise.ufl.edu/~weijenhs/TVC_model (backup page)

Contact email :
wjhsu@ufl.edu

Relevant publications:

- [1] Wei-jen Hsu, Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Ahmed Helmy, "Modeling Time-variant User Mobility in Wireless Mobile Networks," *INFOCOM* 2007, May 2007.

Acknowledgement:
This work is supported by NSF awards CNS-0520017 and career award 0134650.

Content at a Glance

I. Overview

The Time-variant Community Mobility Model [1] is a model proposed to capture the realistic mobility characteristics observed from wireless LAN user traces. Specifically, communities (the area that a user visits with high probability) are used to model the highly *skewed location visiting preferences* and time periods with different mobility parameters are used to model *periodical re-appearance* of a node at the same location. These two properties have been observed from multiple WLAN traces and serve as the motivation of the proposal of such a mobility model.

Although the model was proposed as a way to capture user mobility characteristics observed from WLANs, its applicability extends much beyond WLANs. It has plenty of parameters to produce any location preferences and time period structure in user mobility. We consider it as a flexible mobility model suitable for simulation of mobile networks, such as MANET or delay-tolerant networks.

The details of the model are described in [1] and shall not be reproduced in this manual. The focus of the manual is on how to use the tool to generate a mobility trace of nodes, and how to simulate the hitting time and the meeting time with the tool.

II. Usage of the tool

The code provides multiple executable files with following functionalities. In all of these executables, the parameters for the mobile node(s) are defined in Node.h and Node.cpp (details later). To generate the executables, use the following "make" commands:

Make HT – The executable "HT" simulates the hitting time (i.e., the average time for a node, starting from the stationary distribution, to move into the transmission range of a fixed, randomly chosen target coordinate (i.e., "hits" the target) in the simulation area.).

Make MT – The executable "MT" simulates the meeting time (i.e., the expected time for two mobile nodes, both starting from the stationary distribution, to move into the transmission range of each other.).

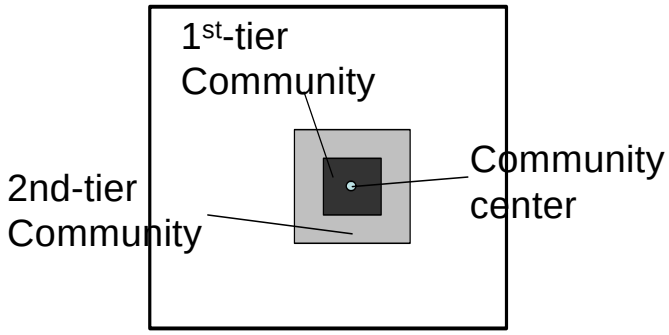
Make MOVELOG – The executable "MOVELOG" prints the movement traces for a pre-defined number of nodes (defined in node_loc_log.cpp as the constant NODES) for a pre-defined time duration (defined in node_loc_log.cpp as the constant END_TIME) in the folder "move_trace/". Trace for each node is printed in separated files. The files are named by "Node<node_id>". The names of files can be changed within the code node_loc_log.cpp. There are two options for the format of the movement traces of the nodes: (1) (t, x, y) format, which prints the time and location of the node for every location update interval (currently set to 0.1 seconds, controllable), or (2) ns2-simulator compatible format.

III. Set up of the parameters

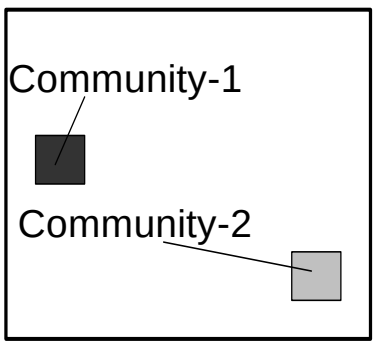
The most involved part of the mobility tool is to set up the parameters for each node. The model provides adequate flexibility to set up the parameters, and in [1] we only used a small set of such capability. In this and the following sections we explain how to set up all the parameters in detail. If you prefer scenarios with pre-set parameters, we provide several files with different parameters we used to match the model with the mobility characteristics from multiple WLAN traces in [1]. See section VII for how to plug in the pre-set parameters.

The most important aspect of the parameter setup is to set the communities. There are three ways to set up the communities:

(1) Multi-tier community: In this setup, each node has a single community in each time period. To model the fine-grained behavior that we move in close vicinity of our frequently visited area (i.e. the community) more often than roam far away from the community, in this setup the community is defined by its center and multiple "tiers" of areas of increasing size and decreasing probability to visit, as illustrated below. The center of the community is chosen uniformly random from the simulation field.



(2). Multiple randomly placed communities: In this setup, each node has multiple communities in each time period, instantiating multiple frequently visited areas. The communities are of the same sizes and randomly chosen from the whole simulation field, as illustrated below. The location of each community is decided independently of the others. Note that it is possible for communities of the same node to overlap, since they are chosen independently at random.



Note that in both (1) and (2) above, if the community is chosen too close to the simulation field boundary (we instantiate the communities by choosing the center at random and then expand the same edge length towards four directions), it will be cut off at the end of the simulation field.

(3). Configuration files: This method allows the user full control of how each node in the simulation is set up. It allows the full freedom to set the mobility parameters for each and every node separately. With this option, one can achieve various scenarios. For example, one can instantiate a few shared communities in the simulation field and each node visits them with different preferences. Alternatively, one can construct a complex scenario, in which sizes, numbers, and preferences of the communities for each node can be different. The format of the configuration files will be introduced in section V after the parameters of the model is discussed in the next section.

IV. Parameters of the time-variant community model

In this section the parameters of the model are explained. If one chooses options (1) Multi-tier community or (2) Multiple randomly placed communities, mobility parameters for each node are the same and defined in Node.h and Node.cpp (hard-coded constants or variables). If one chooses option (3) configuration files, the parameters are fed into the codes through parameter files.

The parameters defined in the file Node.h are:

#define USE_CONFIG_FILE 0 //if config files are used
//0 means configuration files are not used. In this case all nodes follow the same mobility parameters defined in Node.h and Node.cpp (the parameter values are hard-coded as constants and static variables in the code, see below). 1 means configuration files are used. If the configuration files are used, all the "static" variables below will be modified by the configuration files for each node, and hence the values shown in the code will NOT be used.

#define TIERED_COMMUNITY 0 //if the communities are tiered or randomly assigned
//0 means option (2) multiple randomly placed communities is taken, 1 means option (1) Multi-tier community is taken

#define TORUS_BOUNDARY 1
//0 means the boundaries of the communities are reflective, 1 means the boundaries are torus boundary. We use torus boundaries in [1] for its mathematical tractability, but reflective boundaries are more realistic.

#define TRACE_FORMAT 1 //0: (t x y) format, 1: NS-2 format
// 0 means the movement trace generated by node_loc_log.cpp follows the format of (t x y). Every sampling period (defined by TIME_STEP in Node.cpp) the current node location is printed in the trace file. 1 means the movement trace generated by node_loc_log.cpp follows the NS-2 mobility trace format (i.e., using the *setdest* utility in NS-2 to set up the movement epochs). Note that since in the model, the nodes follow straight-line movements so we only need to generate one line in the NS-2 mobility trace for each epoch (unless it hits the boundary to reflect off or "warp" to the other end of the community. If that happens, a single epoch in the mobility model will be broken into "segments" in the NS-2 movement trace format.).

#define PERIOD 2 //number of unique time periods
//number of unique time periods in the simulation – This only defines the number of unique "periods" in which parameters are set separately. The structure of putting these time periods together is defined later. See (**) below.

#define COMMTIER 3 //number of community tiers (or communities)
// Number of community tiers if option (1) Multi-tier community is taken. In this case, the last tier community (the third tier in this case) is always the whole simulation area. The same parameter is used to define number of communities if option (2) multiple randomly placed communities is taken. In this case, there will be 2 communities and the third community is again the whole simulation field. If the user wishes to create a scenario in which the node never roams about the whole simulation field (i.e. no roaming epochs), it can only be achieved by using parameter files (option (3)). Options (1) and (2) by default includes the whole simulation field as the last community.

#define STRUCTURE 100 //longest structure you can have
// This is the longest possible string for the time-period structure. See (**) below.
#define XDIM 1200.0 //size of simulation area
#define YDIM 1200.0 //size of simulation area
// The X and Y dimension for the simulation area
#define TIER_LENGTH 80.0 // 1/2 of edge length of communities
// (Descriptions modified. The manual was wrong here for 1.0-beta version)
// If option (1) Multi-tier community is taken, the edge length of each additional tier of community is calculated by the following: For the first-tier community has edge length 2xTIER_LENGTH (i.e. 160 in this example) – the boundaries are 80 units away from the community center in each direction (up, down, left, and right). From the second-tier and up the community has edges (2k+1)xTIER_LENGTH units away from the center in each direction, where k is (tier-number-1) (i.e. (2x1+1)x80 = 240 units in this example for the second-tier community). The boundaries are 240 units away from the community center in each direction (up, down, left, and right), and that makes the edge length of the second-tier community 480. If option (2) multiple randomly placed communities is taken, each community has edge length of TIER_LENGTHx2 (i.e., 160 units in this case).

#define TIME_STEP 0.1
// The time interval at which the location of nodes are updated. Note that if the communication range of nodes gets smaller, or the average speed gets higher, one will need more frequent location updates to maintain the same simulation accuracy (for the hitting time and the meeting time), since the node locations are updated only at these time steps, and things between the location updates are not “visible” in discrete time simulators. Of course, the simulation would slow down as one reduces the time step.

static double state_prob[PERIOD][COMMTIER] = {{0.8,0.15,0.05},{0.7,0.20,0.10}};
//This parameter defines the probability of choosing an epoch in each (tier) of the communities when the previous epoch finishes. The parameter is set for each time period and each community separately. The first bracket {0.8,0.15,0.05} implies the node has an epoch in tier-one community, tier-two community, and the whole simulation area for probability 0.8, 0.15, 0.05, respectively, during time period 0. The second bracket {0.7,0.20,0.10} defines the probability for time period 1.
// Note this is *different* from how we define the mobility model in [1]. In [1] when we used a single community in each of the time period, it was OK (not too complex) to use the Markov chain model to describe the probabilities of choosing the next epoch type (i.e., in which community) based on the current epoch type. When we extend the model to multiple (tiers of) communities, we think the Markov model would be too complex to specify (e.g., if we have 6 tiers of communities, we need 6x6 = 36 state changing probabilities in the Markov model. There is no theoretical problem in that, but creating such a model is too cumbersome.). So we change to this simplified version. The downside, however, is the lost of capability to instantiate a model

that is likely to take several consecutive epochs in a community and then change state to another community.

static double time_period_dur[PERIOD]={5760,2880};
//This is the length for each time periods. Time period 0 has length of 5760 and time period 1 has length of 2880. When a new time period starts, the users are going to follow the mobility parameters defined for that time period.

static int time_period_structure[STRUCTURE]={0,1};
static int number_of_item_in_structure = 2;
// (**) This array defines how different time periods are put into a repetitive structure. The example here {0,1} means the simulation starts with a time period type 0, followed by a time period type 1, and then the structure repeats itself. The integer number_of_item_in_structure specifies the number of items in the string.
// Such facility can be used to produce interesting repetitive time period structure. One example we mentioned in [1] is the following: we use three distinct time periods to model weekday work hour, weekend day time, and night time separately. To do this, one could define time period 0 as the night time, time period 1 as the weekday work hour, and time period 2 as the weekend day time. Now we need to specify time_period_structure[STRUCTURE]={1,0,1,0,1,0,1,0,1,0,2,0,2,0} and number_of_item_in_structure = 14 to implement the time period structure of a week with five working days, and the same structure will repeat itself. Note the constant STRUCTURE defined above must be greater than number of entries in the string.

static double pause_max[PERIOD][COMMTIER]={{50,20,15},{30,20,30}};
// The maximum pause time after an epoch at various community tier and time period. We assume that the pause time is drawn uniformly from [0, pause_max].

static double l_avg[PERIOD][COMMTIER] = {{140,600,1500},{200,500,1600}};
// The mean length (movement distance) of the epochs at various community tier and time period. The actual length is drawn from an exponential distribution with the same mean.

static double vmin[PERIOD][COMMTIER] = {{4.999,4.999,4.999},{4.999,4.999,4.999}};
static double vmax[PERIOD][COMMTIER] = {{15.001, 15.001, 15.001},{15.001, 15.001, 15.001}};
static double vavg[PERIOD][COMMTIER] = {{10.0, 10.0, 10.0},{10.0, 10.0, 10.0}};
// The minimum, maximum, and average of the nodal speed. We assume the movement speed for each epoch to be drawn independently from the uniform distribution [VMIN, VMAX]. Currently we set the movement speed to be the same regardless of the community the node resides in and the time period, but this can be changed as needs arise in the specific scenario to model.

Some more parameters in other cpp files:
(In ht.cpp and mt.cpp)
#define COMM_RANGE 60
  //the communication range of the nodes

(In node_loc_log.cpp)
#define END_TIME 1000000.0
  //The time duration for which to generate the movement log traces
#define NODES 2
  //The number of nodes to generate the trace for

V. Using Parameter Files

In this section we describe in details the format of the parameter files. If the parameter files are used, they must be placed in the directory “node_parameter_files/”, and with filenames “node<node_id>”. Also note that if this option is chosen, you must prepare a separate parameter file for each and every node in the simulation.

Each of the files should follow the following format:
Line1: community visiting probabilities
Line2: number of items in time period structure
Line3: time period structure
Line4: max pause times
Line5: average epoch lengths
Line6: location of community 0 in time period 0
Line7: location of community 1 in time period 0
…
Line(5+k): location of community k-1 in time period 0 (if there are k communities in time period 0)
Line(6+k): location of community 0 in time period 1
….
Line(5+2k): location of community k-1 in time period 1
…. Repeat until all communities in all time periods are described.
Line(6+kt): minimum movement speeds
Line(7+kt): maximum movement speeds
Line(8+kt): average movement speeds

(Assuming we have k communities in each of the t time periods)
In line 1, the probabilities are listed in the following order: <prob_visit_comm_0_timeperiod_0> <prob_visit_comm_1_timeperiod_0> <prob_visit_comm_2_timeperiod_0> … <prob_visit_comm_(k-1)_timeperiod_0> <prob_visit_comm_0_timeperiod_1> ……. <prob_visit_comm_(k-1)_timeperiod_(t-1)> Each of the parameter is separated by space.

In line 2, the number of items in time period structure should be specified as explained at (**) above. In line 3 the repetitive pattern for the time period structure is given, with each entry separated with spaces. The one-week example above at (**) would result in the following 2 lines:
(Line 2) 14
(Line 3) 1 0 1 0 1 0 1 0 1 0 2 0 2 0

In line 4 and 5, the user should specify the maximum pause time and average epoch length in the same order as in line 1.

From line 6, each line is used to specify a community. The community is specified by four numbers separated by spaces: <left_boundary> <bottom_boundry> <right_boundray> <top_boundary>.

In the last three lines we specify the minimum, maximum, and average movement speed of the node in the same order as in line 1.

Currently the sanity checks for the entered parameters are *not* implemented so it is the user's responsibility to make sure the parameters are typed in correctly (i.e., there is no inconsistencies between the parameters, for example the bottom_boundary for a community is larger than the top_boundary).

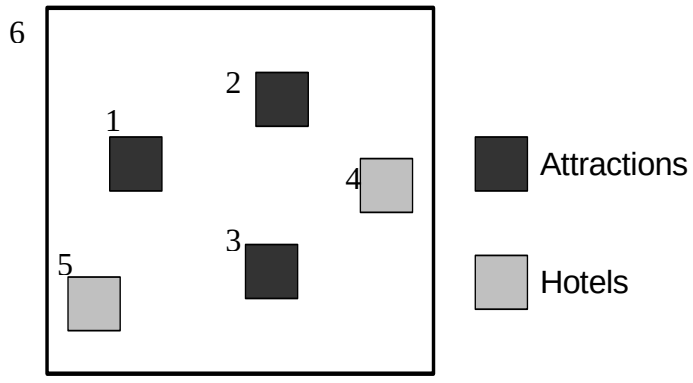
Note that with the configure files, the two constants PERIOD and COMMTIER defined in Node.h can be considered as "maximum" number of time periods and community tiers. One can set some of the nodes without using all of the allocated time periods or community numbers, thereby allowing different nodes to have different numbers of communities, or even different time period structures. For example, if in the Node.h, PERIOD and COMMTIER are set as 3 and 3, respectively, it is possible to set node A's location visiting probabilities as (in line 1 for the node) 0.7 0.3 0 0.6 0.4 0 0.85 0.15 0. In this case, the node never visits its third community, and this is equivalent to the scenario when this node has only two communities in each time period, while it does not prevent other nodes from having three communities.

Similar trick can be played to the time period structure. Although we defined three time period types, it is not necessary that all of them appear in the repetitive structure. Using the same example, in line 3, if we specify 0 1 0 1 1 0, then time period type 2 would never be used for the user.

However, be warned to exercise caution with these parameters. A special note is that although some of the communities or time periods are not used for the node, they still hold their places in all the parameter setup and the community setup lines. The user needs to put in some "dummy values" in their places so the files can be parsed correctly. I recommend to use 0, although it does not really matter for lines other than 1 (in line 1, if a community should never be visited, you *have to* put in 0 for the visiting probability).

Also note with the freedom of setting all the community boundaries by hand, potentially one can set up some of the nodes so they never roam about the whole simulation field. Just hand-pick all of its communities to be smaller than the simulation field.

(An example here) For example, one might be interested in the simulation of visitor mobility and the potential to form a mobility-assisted routing network in a national park as shown below. There are 2 hotels and 3 "major attractions" in the park, as illustrated below. The assumptions are that (1) most visitors go to the attractions during the day and go to the hotels during the night. (2) Some visitors (i.e., the "back-country hikers") will go to the attractions during the day but camp anywhere in the park during the night.



To set up the mobility model, one would need 6 communities (2 hotels+3 attractions+ the whole park. They are identified by the numbers in the figure) and 2 time periods (day-time and night-time). So one would set:

#define USE_CONFIG_FILE 1 //if config files are used
#define TORUS_BOUNDARY 0
#define PERIOD 2 //number of unique time periods
#define COMMTIER 6

For typical visitors who stay in a hotel, they are likely to visit the major attractions with high probability during the day, and stay at the hotel they checked in during the night. So their location visiting probability (line 1 in the config file) is something like
0.3 0.3 0.3 0.02 0 0.08 0.1 0.1 0.1 0.6 0 0.1
with the assumption they go to the attractions equal likely (0.3 probability during the day), and they stay in at the hotel with a small probability (0.02, assuming that they stay at hotel 1), and roam the park with probability 0.08. During the night, with some probability they will stay around the attractions (0.1) or roam around, but most likely they will stay at the hotel (with 0.6 probability). We could set some users to stay at hotel 1 (location 1), and other users to stay at hotel 2 (location 5).
For the hikers, if we assume that they are also likely to visit the major attractions, but they would prefer to walk about the park more than the typical visitors and never go to the hotels, we can set the probabilities as
0.2 0.2 0.2 0 0 0.4 0.1 0.1 0.1 0 0 0.7

The second and the third line of the users would be
2
0 1
to indicate that there are 2 types of time periods and they occur alternatively.

Line 4, max pause times, would be similar to
100.0 100.0 400.0 10.0 0 0 50.0 50.0 150.0 2000.0 0 0
for visitors that use the hotels with the assumption that (1) attraction 3 takes more time to check out than attractions 1 and 2. (2) During the day the visitors only go to the hotel for short stays (use the restrooms, take some short breaks), (3) These visitors do not stop on the road when they roam around the park, (4) During the night the visitors go to the attractions only for some “quick look” so the pause duration is shorter, (5) the pause duration at the hotel is long during the night (once they go back, they are not likely to come out again).
For the hikers it can be set differently, similar to
100.0 100.0 400.0 0 0 50.0 50.0 50.0 150.0 0 0 2000.0
to indicate that they don’t go to the hotels, but find a random place in the park to stay during the night.

In line 5, epoch length should be set with joint consideration of pause time in line 4. For the attractions the visitors may take some time to walk around the area, so the average epoch length should be non-zero and (average epoch length / average movement speed + average pause time) should be the approximate duration a visitor would spend at a location.

Starting from line 6, the community location setup should be the same for all nodes since they are fixed attractions/hotels.

For the movement speed, we can set it higher for typical visitors in the roaming state (i.e., when they move about the whole park) assuming they use cars for long range movement, but set the speed low for local movement around the attractions and the hotel. For the hiker, since they always move on foot, so the speed could be set the same regardless of the state.

VI. Know Limitations and Planned Modifications

VI. 1. The “hopping” behavior

Currently, if a node chooses the next epoch to be in a certain community, while it does not currently reside in the community at the end of the previous epoch, we move the node instantly into the community (place it at a random point within the community). This behavior causes the node to “hop” between places. If one uses the <t, x, y> trace it should be no problem as the trace contains the location of nodes at discrete points in time. The hopping behavior is reflected by nodes showing up at the desired community at the next “t”.
In the ns2-compatible trace format, we implement this by moving the node from its current location to the desired community with high speed (i.e., in 0.000001 seconds) since ns-2 does not allow “hopping” behavior of nodes – movement must be continuous in their simulation framework. This should not be a huge problem as the “zooming across simulation field” makes little communication possible, if any at all, as the contact time between the fast-moving node (to replace the “hop”) and other nodes under normal movement condition should be very short. However, if one is collecting some encounter statistics in ns-2, we cannot guarantee complete correctness with this approximation.

We currently have no plan to change this aspect of the model.

VI. 2. The automatic generation of setup files

We are interested in the problem of setting up the mobility model to instantiate a certain relationship between nodes. This will be done through the configuration files. We will provide a high level “wrapper” script in which one can use several parameters to describe the relationships between the nodes. When we have some result, we will distribute the script too.

This will probably take some time.

VII. Using Pre-configured Parameters

In [1] we have shown a complex community model with matching mobility characteristics with the MIT trace [2]. The parameters we used in the paper is listed in the file MIT_para.txt in our tar file for the program code. This file is simply a copy of Node.h file with different parameters. To use this set of parameters, simply rename MIT_para.txt as Node.h and you are ready.

We have also achieved matching of mobility with the USC trace [3] and the Dartmouth trace [4]. The parameters are kept in the files USC_para.txt and Dart_para.txt.

References

[1] Wei-jen Hsu, Thrasyvoulos Spyropoulos, Konstantinos Psounis, and Ahmed Helmy, “Modeling Time-variant User Mobility in Wireless Mobile Networks,” *INFOCOM* 2007, May 2007.

[2] M. Balazinska and P. Castro, “"Characterizing Mobility and Network Usage in a Corporate Wireless Local-Area Network," In Proceedings of MobiSys 2003, pp. 303-316, May 2003.

[3] W. Hsu and A. Helmy, "On Important Aspects of Modeling User Associations in Wireless LAN Traces," In Proceedings of the Second International Workshop On Wireless Network Measurement, Apr. 2006.

[4] T. Henderson, D. Kotz and I. Abyzov, “"The Changing Usage of a Mature Campus-wide Wireless Network,”" in Proceedings of ACM MobiCom 2004, September 2004.