A Reinforcement Learning Approach to Optimize Downloads Over Mobile Networks

Jayashree Mohan*, Angad Vittal*, K Chandrasekaran* and Bhaskar Krishnamachari†
*Department of Computer Science,National Institute of Technology Karnataka, Surathkal, India
†Department of Electrical Engineering, University of Southern California, Los Angeles, USA
bkrishna@usc.edu

***Abstract*—Dedicated Short Range Communication is attracting a lot of interest these days due to its utility in vehicular safety applications, intelligent transportation system and infotainment applications. Such vehicular networks are characterized by the highly dynamic changes in topology, no significant power constraints and ephemeral links. Considering an interaction between the client and server nodes that last for a random duration of time, an important question is to maximize the amount of useful content downloaded by the client, either in a single request phase, or iteratively in multiple phases. The aim of this work is to propose and investigate a multiphase request model using Markov Decision Process and compare its efficiency against a single phase version. We show that a multiphase request protocol performs better than single phase protocol.**

***Keywords*—Mobile Networks, link layer, optimization, downloads.**

I. Introduction

The tremendous advancements in the IEEE 802.11p, to provide Wireless Access in Vehicular Environment [1] and the increasing use of Dedicated Short Range Communications (DSRC), has led to a growing interest in the field of Inter-vehicular Networking. This emerging technology not only finds use in safety applications, but also provides various other services [2], [3] like email, audio or video sharing between vehicles etc.

Managing efficient communication between vehicles is one of the most fundamental problems in this domain. Maximizing the files downloaded by the client from the server requires optimization at the link layer while utilizing the statistical information about the duration of encounter between the vehicles. Owing to the high switching cost for the client to make multiple requests, a pull based single phase request protocol—MERLIN (Maximum Expected download over Random LINks) [4] was proposed and proved to be optimal. However there are scenarios where such switching costs are considerably low, in which cases generating multiple requests by iterating over a sequence of optimal single phase requests lead to an overall improvement in the performance. The main contribution of our work is the formulation of this problem as a Markov Decision Process (MDP) and show an increased performance over a single phase request version of the protocol.

This work was supported in part by the US National Science Foundation via grant CNS-1217260.

Consider the problem of optimizing downloads in an iterative manner. Given the percentage of files already available at the client, it is time consuming and inefficient to enumerate all the possible client vectors and derive an optimal request plan for each vector composition. Hence the idea is to solve the problem using a Reinforcement Learning approach. Reinforcement learning [5] is learning what to do and how to map situations to actions, in any given environment so as to maximize a numerical reward. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them.

One of the important assumptions in this work is that, the server does not necessarily possess all the files in the repository at the start of encounter, because the server itself is obtaining content through intermittent downloads as in the MERLIN protocol [4]. Similarly, we also assume that the client has downloaded a subset of files from its previous encounters. As assumed in the MERLIN protocol, the distribution of encounter duration is assumed to be known or estimated *a priori* based on historical as well as real-time measurements [4].

In this work, we model each state using parameters like ranges yet to be requested, time remaining till the encounter ends and percentage of files present at the server. While the number of ranges requested at each state specifies the action, a reward equalling the amount of effective data transferred is received. The goal is to find an optimal sequence of ranges to be requested by the client, such that the reward-effective data transfer is maximized. The choice of such a reinforcement learning approach is justified by the fact that, it is impractical to enumerate all possible permutations of files at the client and server and search exhaustively for the best sequence of ranges to be requested, as it turns out to be an exponentially large set that would be computationally intractable for a large repository.

The rest of this paper is organized as follows. Section II discusses some related work. In section III, we provide a brief insight into the MERLIN protocol and the optimization goal for our problem. In section IV, we model a MDP for the problem statement and in section V, we illustrate this model using a simple example. Section VI presents the results in comparison with the single phase protocol from [4] and the baseline push-based approach. The conclusion and possible future work has been described in section VII.

II. Related Works

Content sharing in intermittently connected mobile networks [6] has received quite of lot of attention in the recent times. Several techniques like collaborative content sharing [7] and popularity aware content distribution [8] have been explored. Recent works have also investigated various strategies for content dissemination in such networks including the use of network coding [9] which effectively reduces the file download time and coded storage [10] wherein the use of erasure codes in distributed storage for file sharing speeds up the download of large files.

Such works on content dissemination in mobile networks assume the existence of an efficient link layer which can effectively transfer packets within the short encounters but less attention has been given to the design of such a link transfer mechanism. The MERLIN protocol [4], an efficient single-phase download request protocol for random short-duration communication links, addresses this issue. In this paper we relax the assumptions made by the MERLIN protocol, mainly that the requests take place in a single phase and show how a multi-phase request protocol performs better.

There has been interest in applying reinforcement learning techniques for optimization problems in the area of mobile networking to improve network performance. Reinforcement learning methods have also been used in cellular networks, to find dynamic channel allocation policies that are better than heuristic solutions [11]. Some prior work that uses Reinforcement learning in routing protocols include, the use of Q-Learning algorithm [12] in AODV which makes it more responsive to network topology changes, and the use of collaborative reinforcement learning for MANET routing. Such an approach allows the system to manage higher loads and we use this idea to formulate an iterative request pattern for content downloads in mobile networks.

Having investigated the single phase file request protocol [4] which performs with around 70% efficiency, this work investigates the performance of multiphase model using a reinforcement learning approach.

III. Protocol Overview and Optimization Goal

The MERLIN protocol [4] is a provably optimal, single phase file request protocol that takes into account, the distribution of encounter duration between the client and server nodes (F_T), the number of files available at the server (n_S) and the vector denoting the available and unavailable files at the client (V_C).

A brief overview of how MERLIN protocol works is provided in Figure 1.

1) The server periodically broadcasts a beacon message containing the number of files it contains (n_S), which the prospective client uses to identify its potential server.
2) The client sorts the contiguous ranges of unavailable files in decreasing order—call it the utility function U_{V_C}. The client must request R_b continuous ranges of needed files, which is the variable to be optimized.

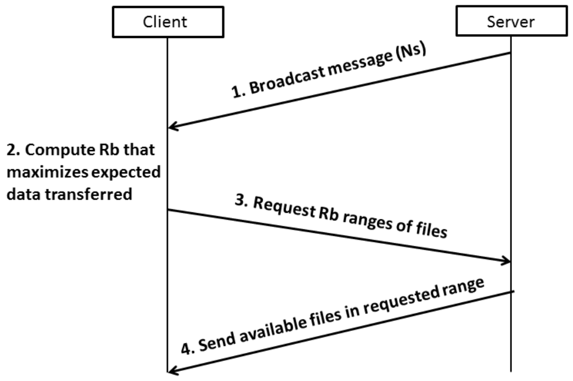
Client
Server
1. Broadcast message (Ns)
2. Compute Rb that maximizes expected data transferred
3. Request Rb ranges of files
4. Send available files in requested range

Fig. 1. Communication flow in MERLIN protocol

If d_{r1} is the data received when the client receives all of the files within the R_b requested ranges which the server has, and d_{r2} is the amount of data received when only a part of the requested files are received, either because the encounter ends during the request or response phase, then R_b is calculated as the maximum expected value of data transferred, as in equation (1)

$$\max_{R_b} E\left[\min(d_{r1}, d_{r2})\right] \tag{1}$$

3) The client requests R_b ranges of files from the server.
4) The server responds with the available files in the R_b requested ranges and the communication ends here.

The performance of MERLIN protocol can be considerably improved if we allow multiple requests by the client. Once the download is completed as described by the MERLIN protocol above, if there is time remaining, the client can again request for another optimized subset of ranges. We associate a penalty with each request phase, which corresponds to the overhead involved in receiving and processing the requests at the server. Given only the percentage of files at the client, it is impractical to enumerate all possible permutations of files at the client and server and search exhaustively for the best solution. Hence we use a reinforcement learning approach towards this problem to find an optimal request policy. Thus the output of the learning algorithm is now the number of ranges to be requested for each request phase.

IV. The solution : Markov Decision Process

In the Reinforcement learning problem, the agent is the learner and decision maker—in this problem, the client mobile node which has to learn over time, the optimal sequence of ranges of file to request. All the other components which affect the decision making and with which the agent interacts comprises the environment. The probability distribution of meeting times of the mobile nodes, the composition and availability percentage of files at the server and client, the client vector, the ranges of files to request from, and the repository size—all of these determine the environment. The agent interacts continually with the environment, selecting appropriate actions which subsequently change the state of the environment accompanied by a numerical reward, which in our case is the effective data transfer that has resulted.

The interaction between the agent and the environment takes place in discrete time steps, t. At each instance of time, the agent receives the representation of the environment's state $S_t \in S$, where S represents the set of all possible states the environment could be in. On this basis, the agent then selects an action $A_t \in A(S_t)$, where $A(S_t)$ is the set of possible actions that could be taken at state S_t, consequently rewarding the agent by R_{t+1} and pushing it to a state S_{t+1}. The mapping from states to the probability of selecting each possible action at a given time step, denotes the policy of the agent π_t, where $\pi(a|s)$ is the probability that $A_t = a$ and $S_t = s$. The agent's goal is to maximize the total reward in the long run.

A. *Formulation of MDP for the Problem Statement*

In our problem, no information about the state of the environment is lost. It is retained and sensed with each action. Since the reinforcement learning task exhibits this Markov Property, we formulate it as a Markov Decision Process (MDP).

1) **Policy (π)** : Ideal sequence of ranges the Client node must request, so as to maximize rewards in the long run.
2) **Goal** : To find the optimal policy π^*
3) **State (S)** : The state of the environment is described by the following parameters:
   - Residual meeting time of client and server
   - Residual ranges of files to select from
   - Percentage availability of files at the server
   - Repository size

   It is important to note that, only the time remaining, ranges to request and percentage of files at the server vary with every state, while the other determinants stay constant throughout the interaction. Hence the total possible number of states, S_{num} is given by (2)

$$S_{num} = t_{max} \cdot R_{max} \cdot sp \tag{2}$$

   - t_{max} represents the maximum time duration for which the mobile nodes interact
   - R_{max} indicates the maximum range from which the client can request files.
   - sp is the percentage of files available at the server.
4) **Action ($A(s)$)** : From any given state s, the number of ranges requested by the client, represents the action.
5) **Reward ($R^a_{SS'}$)**: A numeric value equal to $D^a_{SS'}$, which is the data received by the client, from the server, on transition from state S to S' upon an action a.

6) **Discount (γ)** : $\gamma \in$ [0,1] is the discount factor, which represents the difference in importance between future rewards and present rewards. γ is usually fixed to 0.95.
7) **Transition Probability ($P^a_{SS'}$)** : Equation (3) is the probability that an action a in state s at time t will lead to state s' at time $t+1$.

$$P^a_{SS'} = Pr(s_{t+1} = s' | s_t = s, a_t = a) \tag{3}$$

   This probability can be calculated by estimating the number of hits among the ranges requested by the client, given the repository size. Let N be the number of files present in the ranges requested by the particular action a, and p be the percentage availability of files at the server, then the transition probability can be computed as follows:
   Assuming independence in the request of each file, the transition probability for k hits in a given phase, is simply given by a binomial probability distribution as described in equation (4)

$$P = \binom{N}{k} \cdot p^k \cdot (1-p)^{N-k} \tag{4}$$

   However, the percentage availability of files at the server p changes with every request phase because our range of files of interest keeps narrowing down after each request phase.
8) **Penalty (P)** : A penalty in terms of time is imposed, due to the processing overhead involved in each phase of request.

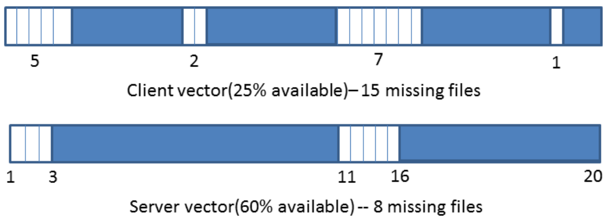
5
2
7
1
Client vector(25% available)–15 missing files
1
3
11
16
20
Server vector(60% available) -- 8 missing files

Fig. 2. A sample client and server vector

Having described each component of the MDP, goal is reached when we maximize the expected discounted sum over a potentially infinite horizon given by equation (5) and in the optimized equation, the solution i.e optimal sequence of actions is as in equation (6)

$$\sum_{t=0}^{\infty} \gamma^t R^{a_t}_{S_t S_{t+1}} \tag{5}$$

$$a_t = \pi(S_t) \tag{6}$$

V. Illustrative Example

Consider the following example that illustrates the optimal policy derivation for a small problem. The parameters used in this problem are

- Repository size= 20 files
- Meeting time = 7 units
- Client availability = 25% i.e 5 files
- Server availability = 60% i.e 12 files
- Penalty = 1 time unit per phase

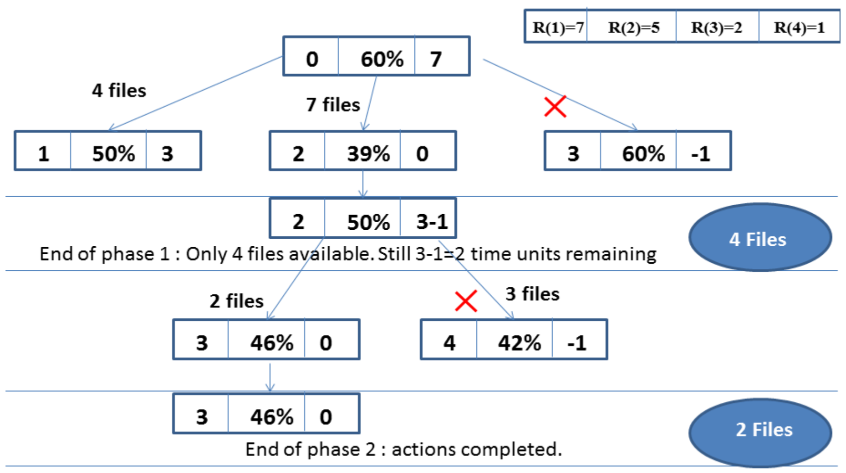

Fig. 3. A sample illustrative example

Figure 2 represents the client and server vector composition before the start of file transfer. The regions marked in blue are the files available and the unshaded regions represent the unavailable files. We can see that the server has files 4–11 and 17–20.

In order to facilitate the requests, we consider contiguous ranges of files that are unavailable at the client and consider a utility vector R, sorted in the decreasing order of size as shown in Figure 3. The reason for requesting ranges in decreasing order has been explained in [4].

The state is represented by three variables, the first being the number of ranges requested so far, the second is the server percentage and the third is the time remaining for interaction.The initial state is represented by the parameters $(0, 60\%, 7)$ meaning, 0 ranges are requested at a 60% server availability, with 7 units of time remaining. At this state, the possible actions are :

1) Request 1 range i.e 7 files
2) Request 2 ranges i.e 7+5=12 files
3) Request 3 ranges i.e 7+5+2=14 files
4) Request 4 ranges i.e 7+5+2+1=15 files

We can see that for actions (3) and (4) the expected values are 8 and 9 respectively, which are greater than the interaction time of 7 units. Hence we can exclude exploring these actions.

Action (1) can lead to 8 possible states as any number of files between 0–7 can be received. The probability of transition to each of these states is given by the probability of number of hits k among the 7 requested files i.e $\binom{7}{k} \cdot (0.6)^k \cdot (0.4)^{7-k}$, where $k \in$ [0,7]. Out of these possible transitions, we see that the probability of transition to the state where 4 files are received is maximum—0.2903. Hence this path is taken. On the other hand if action (2) is performed, then computing transition probabilities in the same way, we see that 7 files are expected to be received. Given these two possible actions in phase 1, the action that yields maximum expected number of files is chosen, and hence action 2 is performed. Performing action (1) instead could lead to the same output (expected reward) eventually, but requires more number of phases, thereby potentially shortening the encounter time due to the penalty imposed on each phase.

Though we expect to receive 7 files by performing action (2), due the server vector composition, we receive only 4 files as the other 3 requested files are unavailable at the server, thereby resulting in a idle time of 3 time units and changing the server percentage to $(12 - 4)/(20 - 4) = 50\%$. After imposing a penalty of 1 time unit, we effectively have 2 units of encounter duration left to utilize in the second phase. Thus the second phase starts with a state $(2, 50\%, 2)$ and the paths are explored in the same way. At the end of phase 2, we exhaust the meeting time, having received 6 files in place of 4, which would have been the result of a single phase request. This action tree is illustrated in Figure 3.

VI. Simulation Based Evaluation

We carried out simulations with the following parameters, using value iteration, to analyze how better the multiphase model works in comparison with a single phase version. The parameters for simulation are as follows

- Mean encounter duration = 50 units
- Overhead to process each file = 1 unit
- Repository size = 100 files
- Client Percentage = 25%
- Penalty per phase = 0.05 unit

When simulated over 100 times for each server availability percentage, for different client vector compositions, Figure 4 illustrates the number of files transferred when multi-phase and single-phase protocols are employed. At lower server percentages, the files transferred are almost similar because at a lower server percentage, higher number of ranges are requested in order to receive the expected number of files. Hence it is possible that there would be no more ranges to request from, even when there is some time remaining. The request for ranges in the decreasing order of size maximizes

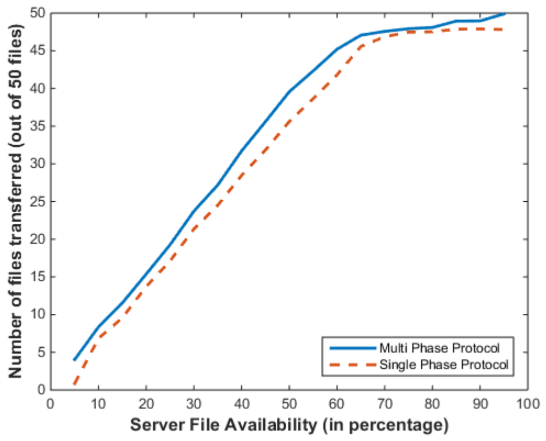

Fig. 4. Comparison of number of files transferred in multiphase with single phase

their total utility within a given request period [4] and hence this pattern of range selection leads to optimal policy.

A. Comparison over Real Trace Simulation

We consider two empirical real trace distributions, one from Beijing taxis and the other from Chicago bus dataset [13], which have an exponential structure as shown in Figure 5. In the simulations below, we compare the performance of multi-phase request protocol with two other methods described below:

1) Single phase MERLIN protocol: This is a pull based protocol in which the request-response takes place in a single phase [4].

2) Push: This algorithm has no request phase from the client and the server sequentially sends out the available files.

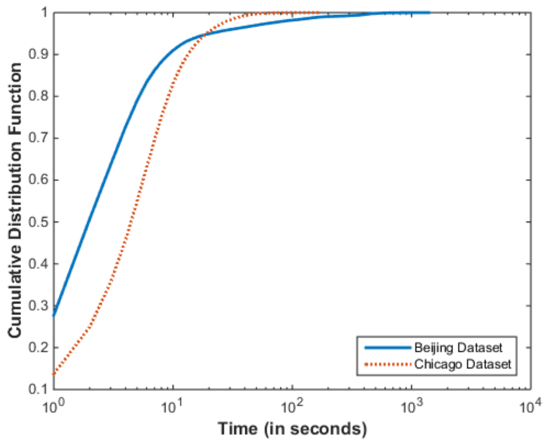

Fig. 5. Cumulative distribution function for Beijing and Chicago dataset

In our simulations, the following are the default set of values

- Repository = 500 files
- 30% client availability
- 200 mean encounter duration in time units (default distribution: exponential)
- 10 file size (in time units)
- 2 header size (time units)
- 2 time needed to describe each range
- Penalty = 5 time units

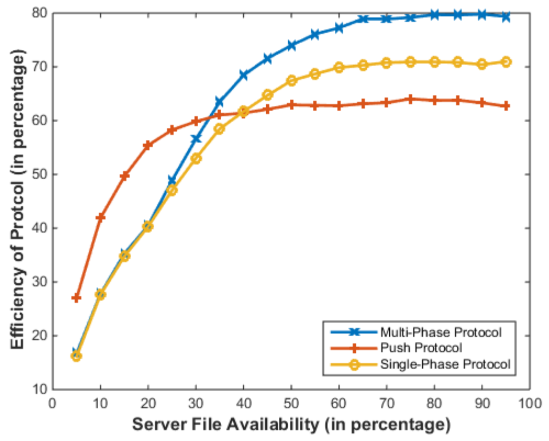

Fig. 6. Comparison of performance of multiphase version versus single phase and server push methods for Beijing vehicular trace

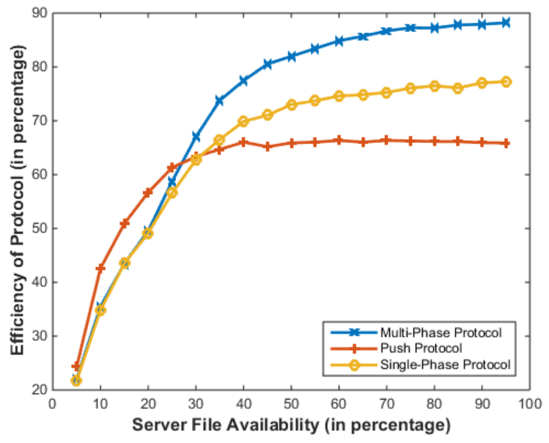

Fig. 7. Comparison of performance of multiphase version versus single phase and server push methods for Chicago vehicular trace

As can be seen from Figure 6 and Figure 7, the efficiency of multi-phase request model is considerably higher than single phase and server push methods, at moderately high server availability percentages because the idle time is utilized effectively in making more requests. However at lower server availability percentages, multi-phase requests are same as single phase, mainly because almost all ranges are requested at lower server percentages, because the expected number of files transferred is low. And it is not better than server push at this end of the spectrum, because it is wise to push the minimal available files rather than wasting resources in pulling them.

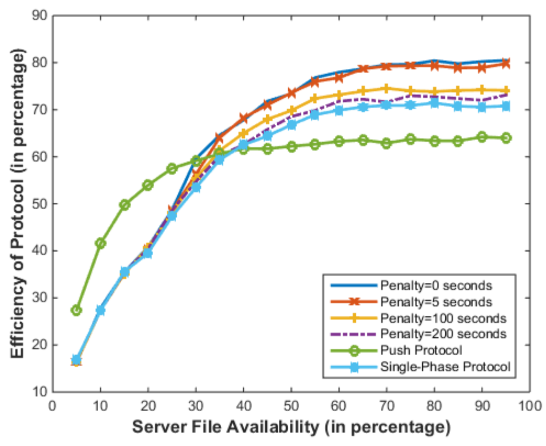

Fig. 8. Comparison of performance of multiphase version for different values of phase penalty

TABLE I. RANGES REQUESTED PER PHASE

Server Percent	Ranges requested in Phase 1	Ranges requested in Phase 2	Ranges requested in Phase 3
5	22		
10	19		
15	19		
20	25		
25	21	3	
30	12		
35	7		
40	11		
45	6		
50	4		
55	7		
60	7	9	1
65	3	6	9
70	4		
75	3	8	12
80	4		
85	5		
90	3	5	11
95	3		

We try to analyze the impact of penalty associated with each request phase on the efficiency. The tradeoff between overhead involved in initiating a new request for each phase and the goodput is as shown in Figure 8. It is easily seen that, as penalty increases the efficiency reduces because the useful time spent in data transfer reduces.

Table I represents an instance from simulation, where the number of ranges requested per phase and the corresponding server percentage is shown. It can be seen that multiple requests come into picture only beyond median ranges of server percentages, for reasons explained previously. The number of ranges requested in each phase initially increases because, in the initial phases when the size of ranges is large (it is sorted) few of them are requested. In the subsequent phases both the time remaining and size of the ranges decreases, but the reduction in range sizes dominates the decrease in time, thereby resulting in the request of higher number of ranges. However, when the decrease in time dominates the reduction in range sizes, we see a fall in the number of requested ranges as in the case of 60% server availability.This request pattern is dependent on client and server vectors.

VII. Conclusion

In this paper we have formulated the problem of optimizing downloads in a mobile network as a Markov Decision Process, using Reinforcement learning. We have compared the performance of this approach to the baseline methods and shown that the multi-phase approach leads to a better performance in comparison to server push and single phase protocols.

One possible direction to investigate further is that, in the case of a vehicular network where multiple vehicles compete for resources and the encounter duration is potentially very short, if a single node keeps requesting for files iteratively over multiple phases, it is blocking access to resources for other vehicles which could have received a few files in the duration this vehicle made its second round of request.

Another direction to be explored is to improve the efficiency at lower server availability, by employing a hybrid scheme, where push or pull is determined for each phase depending on the availability percentages at the client/server respectively.

References

[1] Jiang, Daniel, and Luca Delgrossi. “IEEE 802.11 p : Towards an international standard for wireless access in vehicular environments.” Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE. IEEE, 2008.

[2] Lee, Uichin, Ryan Cheung, and Mario Gerla. “Emerging vehicular applications.” Vehicular Networks: From Theory to Practice, chapter Chapter1 (2009).

[3] Moustafa, Hassnaa, and Yan Zhang. Vehicular networks: techniques, standards, and applications. Auerbach publications, 2009.

[4] Bhargava, Amber, Spencer Congero, Timothy Ferrell, Alex Jones, Leo Linsky, Jayashree Mohan, and Bhaskar Krishnamachari. “Optimizing Downloads over Random Duration Links in Mobile Networks.” In Computer Communication and Networks (ICCCN), 2016 25th International Conference on, pp. 1-9. IEEE, 2016.

[5] Sutton, Richard S., and Andrew G. Barto. “Reinforcement learning: An introduction.” Vol. 1. No. 1. Cambridge: MIT press, 1998.

[6] Abbas Jamalipour, Yaozhou Ma, “Intermittently Connected Mobile Ad Hoc Networks: from Routing to Content Distribution”, Springer, 2011.

[7] Johnson, Mark, Luca De Nardis, and Kannan Ramchandran. “Collaborative content distribution for vehicular ad hoc networks.” Allerton conference on communication, control, and computing. 2006.

[8] Zhang, Yang, Jing Zhao, and Guohong Cao. “Roadcast: a popularity aware content sharing scheme in vanets.” ACM SIGMOBILE Mobile Computing and Communications Review 13.4 (2010): 1-14.

[9] U. Lee, J. Park, J. Yeh, G. Pau, and M. Gerla. “Code torrent: content distribution using network coding in VANET”. In ACM MobiShare, 2006.

[10] Sathiamoorthy, Maheswaran, et al. “Distributed storage codes reduce latency in vehicular networks.” Mobile Computing, IEEE Transactions on 13.9 (2014): 2016-2027.

[11] Singh, Satinder, and Dimitri Bertsekas. “Reinforcement learning for dynamic channel allocation in cellular telephone systems.” Advances in neural information processing systems (1997): 974-980.

[12] Celimuge, W. U., and Kazuya Kumekawa. “Distributed reinforcement learning approach for vehicular ad hoc networks.” IEICE transactions on communications 93.6 (2010): 1431-1442.

[13] Vehicle Trace Datasets : This code was obtained from research conducted by the University of Southern California’s Autonomous Networks Research Group, http://anrg.usc.edu, May 2013.