Impact of Persistent Storage on the DTN Routing Performance

Veeramani Mahendran, Thammana Praveen, and C. Siva Ram Murthy

Department of Computer Science and Engineering
Indian Institute of Technology Madras
Chennai 600036, India
mahendra@cse.iitm.ac.in, praveen.thammana@gmail.com, murthy@iitm.ac.in

Abstract. The *store, carry, and forward* paradigm of the Delay-Tolerant Network (DTN) architecture enables a node to carry messages for a long period of time. This long-term storage is supported by the DTN architecture with the usage of persistent storage; however to the best of our knowledge, the routing/scheduling framework that incorporates support for persistent storage has not been addressed much in the DTN literature. In this paper, we investigate the impact of persistent storage on the routing performance over different buffer scheduling policies. Our extensive simulation studies demonstrate that they exhibit an improvement in delivery ratio, but with a compromise on delivery delay. This shows the pressing need for a new scheduling policy to tap the complete potential of the persistent storage. To this end, we propose a Time in Primary Scheduling (TiPS) policy with two variants (one using local information and the other using global information) that outperforms the contemporary buffer scheduling policies with respect to the persistent storage framework.

Keywords: DTN, routing performance, persistent storage.

1 Introduction

Delay-Tolerant Networks (DTNs) [4], [7] are the challenged networks that are used in highly dynamic environments where end-to-end connectivity is not always possible. DTNs support a wide variety of potential applications that include military systems, disaster management systems, and content dissemination systems involving smart phones.

The message propagation in the traditional packet-switched networks (such as TCP/IP) is governed by *store and forward* paradigm. The reason behind this fact is that, in TCP/IP networks an end-to-end connection is assumed to last longer when compared with the granularity of the packet transfer time over a single link. Hence a packet can be stored or forwarded immediately to the next hop node. Unlike traditional networks, the existence of link in DTN pertains to the physical mobility of the nodes. A message can be transferred (in full or part[1]) to

[1] DTN architecture supports reactive fragmentation, wherein the part of the message transferred till the link breakage is treated as a new message.

L. Bononi et al. (Eds.): ICDCN 2012, LNCS 7129, pp. 513–524, 2012.

the next hop node only when the link becomes available. There is a high chance for a node to carry the message (most of the time) before forwarding it to another node and hence the paradigm is so named as *store, carry, and forward* [4]. The DTN architecture backs up this paradigm with the help of persistent storage in the nodes.

Further, the DTN uses an indigenous custody transfer mechanism that delegates the re-transfer capability of a source node to the other nodes. These nodes that take care of custody transfer (called as custodians) cannot afford to lose the messages that are taken into custody, thereby making the persistent storage a *de facto* requirement in DTN.

The literature in the context of DTN routing considers either infinite or finite storage space (a monolithic storage resource of primary memory) in the system model, while the former is unrealistic and the latter (rather than being a common denominator framework as in contemporary networks) does not represent the DTN architecture in a complete way. The vital role played by the persistent storage (in addition to the primary/internal memory) in the routing is not yet addressed.

In this paper, we consider a 2-level hierarchical storage (with one level being the primary or internal memory, and the other being the persistent or secondary storage) model in the nodes and study the DTN routing performance for this framework[2]. Without loss of generality, we assume that the messages in persistent storage of the node remain dormant during a contact. The contributions of this paper are as follows:

- *Persistent storage gain* - Study the rudimentary gain caused by the persistent storage, in terms of the fraction of additional messages that (are dropped in a primary-only framework or remain dormant in the secondary storage of the persistent framework) can participate in the routing. This gain is quantified by using an end-to-end metric named as Gain due to Persistent Storage (GPS).
- *Persistent storage influence on routing* - Investigate the impact caused by the persistent storage when coupled with the contemporary buffer scheduling policies on the routing protocol performance.
- *Time in Primary Scheduling (TiPS) policy* - Propose a novel buffer scheduling policy that schedules messages in a manner that gives all the messages an equal chance to stay in the primary memory. We evaluate the performance of the policy through extensive simulation.

The rest of this paper is structured as follows: Section 2 presents the related work done in the context of buffer management in DTN. In Sect. 3, we motivate the work by studying the improvement caused by the mere extension of persistent storage to the existing framework. We present the persistent storage framework in Sect. 4 and brief on the routing performance for this framework. In Sect. 5,

[2] Henceforth we treat routing and buffer scheduling framework simply as the framework and, the framework with persistent storage as persistent framework and the existing framework without persistent storage as primary-only framework.

we propose our novel scheduling policy for the persistent storage framework and evaluate its performance. Finally, we conclude the work in Sect. 6 and suggest some future directions in Sect. 7.

2 Related Work

In the context of buffer size of the node, the system models in the DTN literature can be classified as infinite buffer [10] and finite buffer (single primary queue) [8], [11], [12] models. The work in the latter (also called as constrained buffer model) mainly focuses on the buffer scheduling policies (such as drop tail, drop front, random schedule and Global Buffer Scheduling and Drop (GBSD) [8], [11]) and their role in improving the routing performance. Here, we discuss them briefly as follows: (*i*) Drop tail - a new message entering the node if finds the (primary) buffer full would be dropped, (*ii*) Drop front - a new message would replace the old message at the head of the queue, (*iii*) Random - a new message would replace a randomly chosen message in the buffer, and (*iv*) Global knowledge Based Scheduling and Drop (GBSD) [8] - a utility based metric that uses a global information (pertaining to the infected copies of the message) is computed to drop the least utility packet thereby optimizing the delivery delay.

While the research on buffer management policies and routing protocols of DTN are done independently, the authors in [9] integrate them together by migrating (routing) the excess messages to the neighbor nodes that have enough space to carry them. On similar lines, the authors in [5] investigate the fairness model for the DTN node buffer allocation among the active sessions. They propose a buffer efficient routing scheme that provides a fair share of the buffer usage and achieves a better throughput as well.

The authors in [6] are perhaps the first to address the persistent storage in DTN. The incoming messages are categorized based on the traffic flow (especially low and high delay traffic). The primary memory contains two logical queues (Low Delay Traffic (LDT) queue and High Delay Traffic (HDT) queue) to house the corresponding messages. The LDT messages have more priority over HDT messages. The peristent storage is used as a backup to store the excess messages. However, specific characteristics of the messages stored in persistent storage (such as delayed access of messages or dormant messages in an active contact) are not considered. This makes the model semantically equivalent to the infinite buffer model.

3 Motivation

3.1 System Model

In this section, we define the system model in detail. We consider a system of N nodes moving according to Random-WayPoint (RWP) mobility model [3] in a

square terrain. Source and destination nodes are chosen uniformly random with each node being a source for itself and destination for other. The routing protocol under consideration is the epidemic routing protocol with VACCINE recovery scheme [11]. The propagation model used is *TwoRayGround*. The MAC protocol used for all the nodes is 802.11. The simulation settings are shown in Table 1. All graphs are plotted with 95% confidence level. The routing performance metrics under study are as follows:

Table 1. Simulation parameters

Parameter	**Value**
Number of nodes	20
Velocity of the nodes	6m/sec with no pause time
Traffic model	Exponential with $\lambda = \frac{1}{45}$
Terrain size	500m × 500m
Transmission radius	50m
Message TTL	1s to 400s (uniformly random)
Total simulation time	2000s

- Delivery ratio: Fraction of successful packets delivered at the destination nodes.
- Delivery delay: Delay incurred by sending the packets from the source node to the destination node.

In this section, we verify the scope of improvement provided by the persistent framework when compared to the existing primary-only framework. This is non-trivial because both frameworks (primary-only and persistent) come with a price. In primary-only framework, the price is in terms of permanent loss of dropped packets and in persistent framework the price is in terms of secondary messages being dormant during a contact; hence cannot be forwarded (further details regarding this is deferred to Sect. 4). Hence we need an end-to-end metric (called GPS) to study this gain. GPS is computed as follows: Let $\mathbb{U}$ be the set of undelivered messages and $\mathbb{G}$ be the set of total messages generated in the network. $< m, \mathbf{n_a}, \mathbf{n_d}, \mathfrak{t} >$ be a tuple that denotes the fact that the message m (destined to the node $\mathbf{n_d}$) was dropped by its associated node $\mathbf{n_a}$ and the node $\mathbf{n_a}$ has met the node $\mathbf{n_d}$ in time $\mathfrak{t}$ within the lifetime of the message m (*i.e.*, $\mathfrak{t}_{g_m} \leq \mathfrak{t} \leq TTL_m$; where $\mathfrak{t}_{g_m}$ is the message generation time of the message m). Let us define the set $\mathbb{F}$ as follows:

$$\mathbb{F} = \{m :< m, \mathbf{n_a}, \mathbf{n_d}, \mathfrak{t} >, (m \in \mathbb{U}), (\mathfrak{t}_{g_m} \leq \mathfrak{t} \leq TTL_m)\} \tag{1}$$

Now, the effective metric I_p is computed as follows:

$$I_p = \frac{|\mathbb{F}|}{|\mathbb{G}|} \tag{2}$$

where $|.|$ is the cardinality of the set. The effective metric I_p denotes the fraction of dropped messages that would have been delivered otherwise.

In a similar way, for the persistent storage, let us consider a set $\mathbb{F}$ such that $< m, \mathbf{n_a}, \mathbf{n_d}, \mathbf{t} >$ is a tuple that denotes the fact that the message m (destined to the node $\mathbf{n_d}$) was in the secondary storage when its associated node $\mathbf{n_a}$ has met the node $\mathbf{n_d}$ within the message's lifetime and define the corresponding effective metric as I_s. The metric I_s in this case denotes the fraction of messages that have missed their destination nodes while being dormant in the secondary storage of the associated node $\mathbf{n_a}$. The GPS metric is computed as follows:

$$GPS = I_p - I_s \tag{3}$$

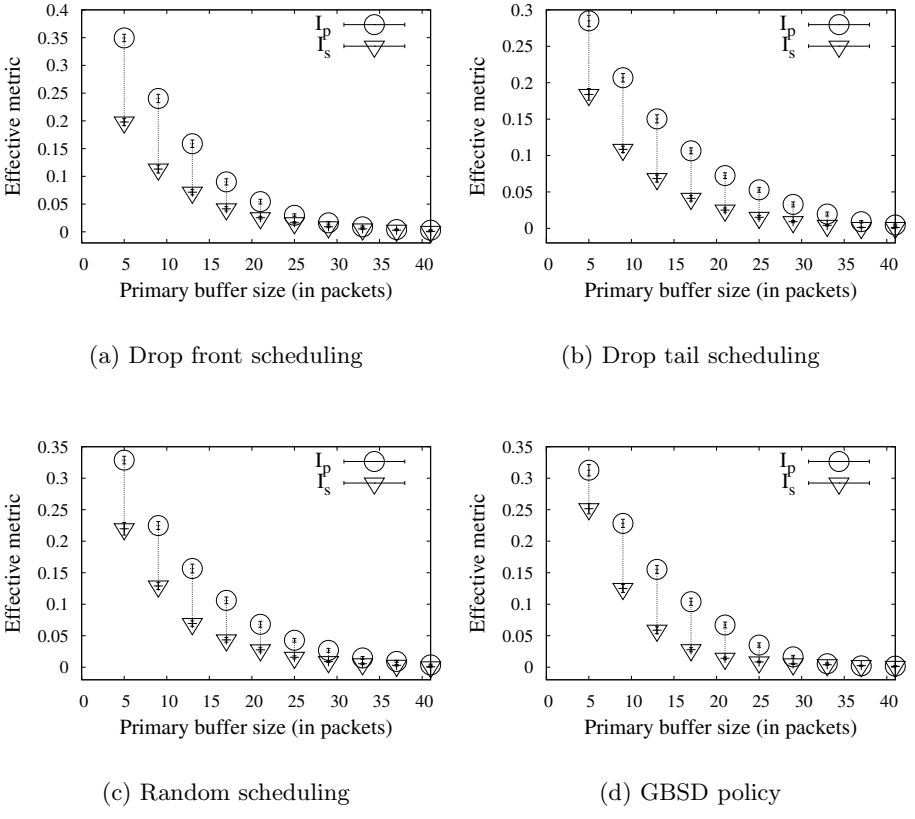
This metric denotes the effective fraction that can be saved in the persistent framework. The effective metrics I_p and I_s are computed for different scheduling policies by varying different primary buffer size and the results are plotted in Fig. 1. The GPS metric is shown in the figure as *vertical lines* connecting the metrics I_p and I_s at each primary buffer size. The GPS is relatively high at low primary buffer sizes and this trend reduces to zero and stays thereon indicating the fact that the primary buffer size is sufficiently large enough to hold all messages; hence, persistent framework has no impact at these values of primary buffer size.

As a real example, consider a DTN application that runs in android (an open source mobile operating system) phones. A nominal cost[3] android phone features a few hundred MB of internal user memory (primary storage) with external (persistent storage) SD card support up to a few GB. Such a system will definitely perform well if the external storage is exploited. Furthermore, the best-in-class average read access speed (about 159 Mbps [2]) of the MicroSD card (persistent storage in smart phones) is still slower than the best-in-class sustained throughput (about 182 Mbps) of the 802.11n WiFi radio [1]. This is evident to have a bi-level storage framework in the node.

4 Persistent Storage Framework

Figure 2 shows the existing primary-only DTN framework extended with the persistent storage (*gray-scale* portion in the figure denotes the add-on part). The persistent storage can be disk storage or an external storage MicroSD card. The data access logic is the data management and access service interface for the persistent storage. The persistent storage add-on enables the buffer scheduling policy to perform an added functionality of inter-message scheduling between primary and secondary queue.

[3] Under $15K$ price range in Indian currency.



(a) Drop front scheduling

(b) Drop tail scheduling

(c) Random scheduling

(d) GBSD policy

Fig. 1. Gain due to Persistent Storage (GPS)

Figure 3 and Fig. 4 depict the routing performance for primary-only and persistent framework over four different scheduling policies. The results show that the persistent storage is fruitful in giving better improvement in terms of delivery ratio across all scheduling policies, but with delivery delay being high. This is obvious as more messages contribute to the delay because of the relatively reduced message drop due to persistent framework (a behavior that is also reflected in Fig. 1). This shows us the scope for devising a new scheduling policy that exploits the persistent storage. From Fig. 3 and Fig. 4 it is clear that the GBSD policy outperforms all other scheduling policies in the persistent framework, henceforth, we would be using GBSD as the reference scheduling policy to compare against the new scheduling policy.

With the persistent storage add-on, the message drops that occur (based on the scheduling policy) at the primary buffer will be backed up in the secondary storage. In a similar way, the messages are pumped in to the primary buffer based on the reverse scheduling policy (For example, in GBSD if a message with least utility is dropped to the secondary storage, then the message with maximum

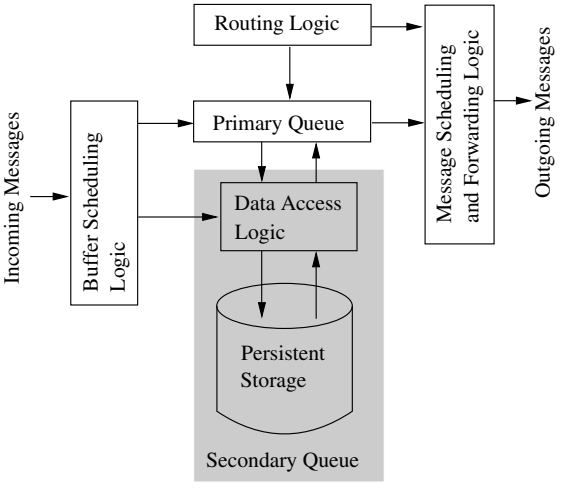
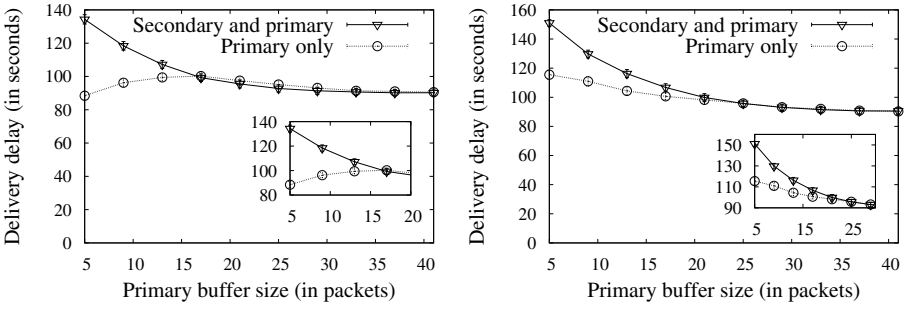
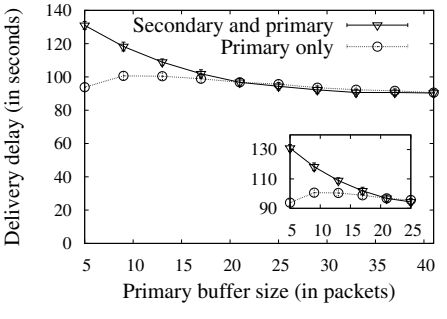


Fig. 2. DTN framework extended with the persistent storage

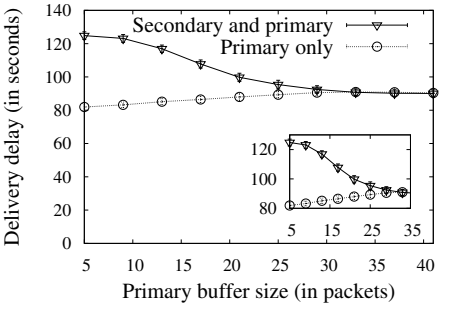


(a) Drop front delay

(b) Drop tail delay

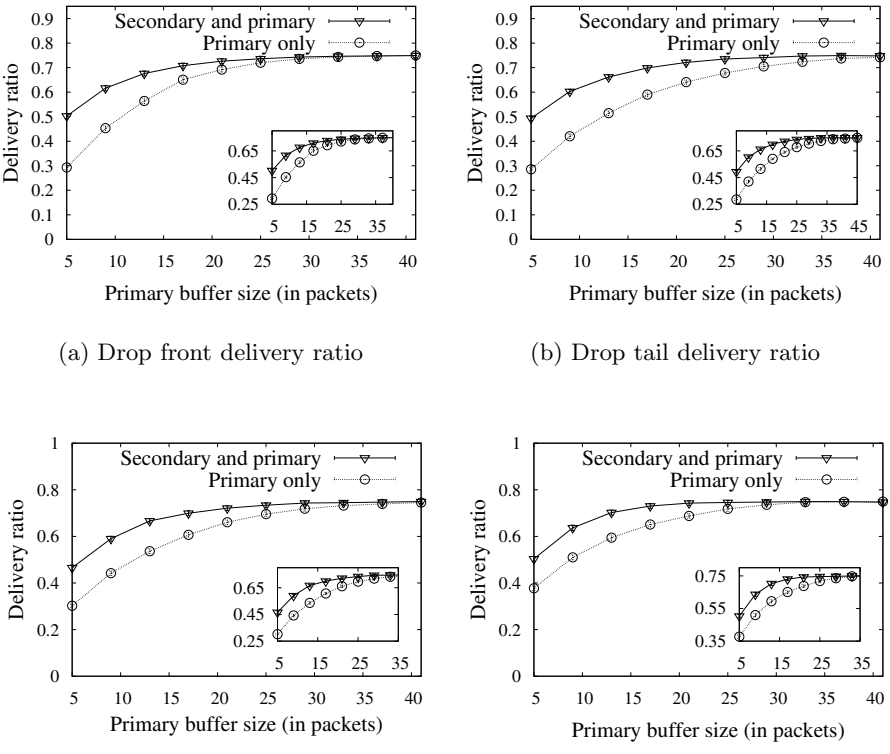


(c) Random delay



(d) GSBD delay

Fig. 3. Delivery delay for different scheduling policies



(a) Drop front delivery ratio

(b) Drop tail delivery ratio

(c) Random delivery ratio

(d) GSBD delivery ratio

Fig. 4. Delivery ratio for different scheduling policies

utility in the secondary storage would be moved to the primary memory). The scheduling is set to happen after every contact made by a node with other nodes. This is done to ensure proper mixing of messages between primary and secondary storage.

Without loss of generality, we assume that the persistent storage is sufficiently large enough to hold all the messages. Any two nodes in a current contact would restrict access only to their messages stored in the primary memory. This assumption is made for the following reasons: (*i*) The time taken to retrieve a message from secondary to primary storage is device dependent and not known. Hence, a message transfer from secondary storage of a node to another node in the same contact is deemed unrealistic, (*ii*) To ensure the bandwidth of the contact to be common across both primary-only and persistent framework under consideration.

5 Time in Primary Scheduling (TiPS)

In this section, we propose a new buffer scheduling policy (TiPS) that exploits the persistent storage. The objective of this scheduling policy is to reduce the

delivery delay while still maintaining the delivery ratio. Hence the rationale behind our scheduling policy is to give all messages equal chance to stay in the primary memory. To do so, we compute the total time spent by a message in the primary memory as follows: Let h_m be the number of hops traveled by a message m thus far to reach the current node $\mathbf{n}$ (h_m is initialized to 0 when the message m is generated). Therefore, the total time spent by a message m in the primary memory T_{p_m} of a node $\mathbf{n}$ is computed as follows:

$$T_{p_m} = \sum_{i=0}^{h_m} T_i \tag{4}$$

where T_i is the time spent by a message in a node it has entered in the i^{th} hop. And this aggregate time is maintained in the header of each message.

The scheduling policy computes T_{p_m} for all messages m in the primary memory and schedules the one with maximum value to the secondary storage. Since this policy uses message-level local information, this variant is referred as Local TiPS (LTiPS). Therefore, the message to be scheduled into the secondary storage in LTiPS is the one that has the utility value given by,

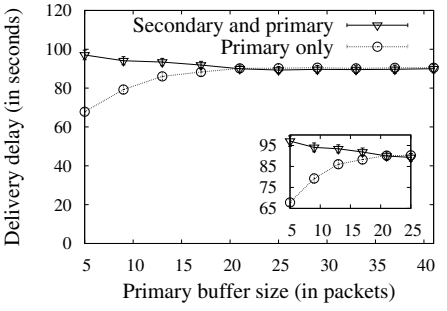
$$\max_{m \in \mathbb{P}} \{T_{p_m}\} \tag{5}$$

where $\mathbb{P}$ is the set of all messages present in the primary memory of the node $\mathbf{n}$. This ensures that the messages that have (so far) spent least time in the primary memory would get high priority to stay there for more time. The more time a message spends in the primary memory the more it replicates the message due to the epidemic routing, hence with more copies in the network the delivery delay of the message will get improved.

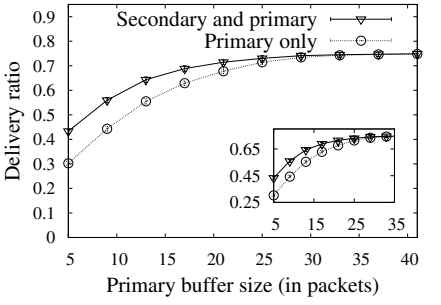
Since one of the existing scheduling policies use the global information [8], to ensure common denominator, we extend the version of TiPS with global information as well (Henceforth the variants are named as LTiPS and GTiPS respectively). In this global variant, the message m with maximum normalized infected copies along with the total time spent in the primary memory will be scheduled to the secondary storage. Therefore, the Global TiPS (GTiPS) is given by the following utility:

$$\max_{m \in \mathbb{P}} \{\frac{S_m}{N} \times T_{p_m}\} \tag{6}$$

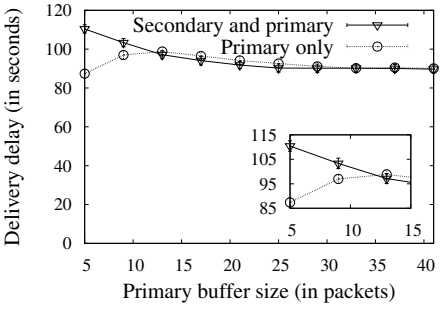
where S_m is the total number of infected copies of the message m present in the primary memory and N is the total number of nodes in the network. The LTiPS and GTiPS scheduling policies were evaluated and their performance is shown in Fig. 5. Their performance trend is similar to that of the existing scheduling policies. From Fig. 3, we observe that the GBSD outperforms when compared to other existing scheduling policies in the persistent framework. Hence we compare the behavior of GBSD with the proposed scheduling policies LTiPS and GTiPS. In terms of delivery delay both LTiPS and GTiPS show significant improvement than GBSD as shown in Fig. 6. The delivery ratio of LTiPS policy



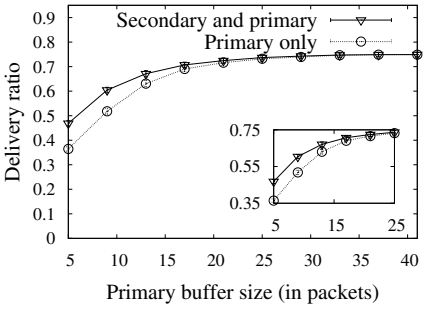
(a) LTiPS Delay



(b) LTiPS Delivery ratio

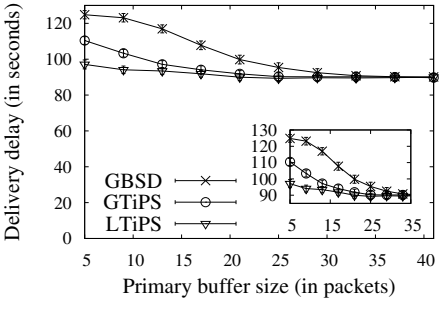


(c) GTiPS Delay

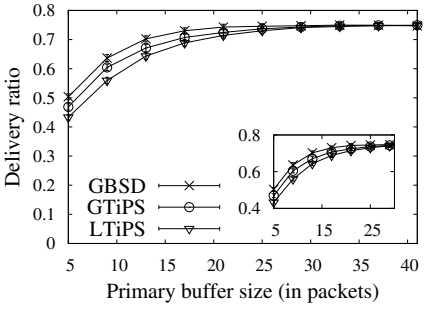


(d) GTiPS Delivery ratio

Fig. 5. Delivery delay and delivery ratio for LTiPS and GTiPS scheduling policies



(a) Delay



(b) Delivery ratio

Fig. 6. Delivery delay and delivery ratio for GBSD, LTiPS, and GTiPS scheduling policies

is not close to that of GBSD. But GTiPS gains the best results in terms of delivery ratio being statistically equivalent to that of GBSD and this is achieved with a significant improvement on the delivery delay. This happens due to the fact both GTiPS policy and GBSD use global information thereby standing on the common platform, whereas LTiPS does its best with the locally available information.

All the performance plots converge beyond some point (say at primary buffer size as 40 messages) as the primary buffer is sufficient enough to hold all the messages for the given system parameters; this behavior is also reflected in Fig.1 as well.

6 Conclusion

In this paper, we have studied the impact of persistent storage of DTN nodes on the routing performance. A systematic way to quantify the rudimentary improvement caused by the persistent storage is proposed using a new metric (GPS). We have also demonstrated that the existing buffer scheduling policies with the persistent storage framework do not perform well in terms of delay, which led us to propose a new buffer scheduling policy (TiPS). This policy when integrated with the persistent storage framework outperforms the existing class of scheduling policies in terms of delay while maintaining the delivery ratio to be statistically equivalent.

7 Future Work

We assumed that the persistent storage to be large enough to hold all messages. The messages cannot stay forever in the node due to their finite lifetime (TTL); thereby bounding the persistent storage to a finite size. Our future focus would be on quantifying the persistent storage size under a given traffic model. We would also look into integrating routing protocol with the scheduling policy in the persistent storage to further improve its performance.

Acknowledgement. This work was supported by the Department of Science and Technology, New Delhi, India.

References

1. Cisco and intel: Collaborative 802.11n leadership and testing, `http://www.cisco.com/en/US/solutions/collateral/ns340/ns394/ns348/ns767/white_paper_c11-492743_v1.pdf`
2. Microsd card performance test results, `http://www.sakoman.com/OMAP/microsd-card-perfomance-test-results.html`
3. Bettstetter, C., Hartenstein, H., Pérez-Costa, X.: Stochastic Properties of the Random Waypoint Mobility Model. Wireless Networks 10, 555–567 (2004)

4. Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., Weiss, H.: RFC 4838, Delay-Tolerant Networking Architecture. IRTF DTN Research Group (2007)
5. Chuah, M.C., Ma, W.B.: Integrated Buffer and Route Management in a DTN with Message Ferry. In: MILCOM 2006: Proceedings of the IEEE Conference on Military Communications, pp. 1–7 (2006)
6. Dimitriou, S., Tsaoussidis, V.: Effective Buffer and Storage Management in DTN Nodes. In: ICUMT 2009: Proceedings of the International Conference on Telecommunications, pp. 1–3 (2009)
7. Fall, K.: A Delay-Tolerant Network Architecture for Challenged Internets. In: SIGCOMM 2003: Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, pp. 27–34 (2003)
8. Krifa, A., Barakat, C., Spyropoulos, T.: An Optimal Joint Scheduling and Drop Policy for Delay Tolerant Networks. In: WoWMoM 2008: Proceedings of the International Symposium on World of Wireless, Mobile, and Multimedia Networks, pp. 1–6 (2008)
9. Seligman, M., Fall, K., Mundur, P.: Storage Routing for DTN Congestion Control: Research Articles. Wireless Communications and Mobile Computing 7, 1183–1196 (2007)
10. Spyropoulos, T., Psounis, K., Raghavendra, C.S.: Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In: WDTN 2005: Proceedings of the ACM SIGCOMM Workshop on Delay-Tolerant Networking, pp. 252–259 (2005)
11. Vahdat, A., Becker, D.: Epidemic Routing for Partially Connected Ad hoc Networks. Tech. Rep. CS-2000-06, Duke University (2000)
12. Xu, B., Wolfson, O., Naiman, C.: Machine Learning in Disruption-Tolerant MANETs. ACM Transactions on Autonomous and Adaptive Systems 4(4), 1–36 (2009)