On the broadcast of segmented messages in dynamic networks

Sandipan Sikdar*, Marcin Bodych†, Rajib Ranjan Maiti‡, Biswajit Paria*
Niloy Ganguly*, Tyll Krueger† Animesh Mukherjee*
*Indian Institute of Technology Kharagpur, Kharagpur, India
†Wroclaw University of Technology, Wroclaw, Poland
‡Instituto di Informatica e Telematica, Pisa, Italy

***Abstract*—This paper makes a systematic attempt to understand the effect of message size on the speed and efficiency of message broadcast. It considers a realistic situation where a single message may be too large to be sent in over a single connection and hence might require to be transmitted in segments. In specific, we look into the $push$ and $pull$ message transfer techniques and investigate in details their effect on broadcast time as well as total number of redundant contacts incurred during the transmission of segmented messages. For such segmentation and a complete graph topology with n nodes, we observe that the time required for broadcast scales as $n^{\frac{k-1}{k}}$ (assuming there are k packets in one message segment) as opposed to $\log n$ in the single message epidemic case $(k = 1)$. In order to improve broadcast time and reduce the number of useless contacts we propose different variants of the $push$ and $pull$ message transfer techniques. In this regard we introduce the concept of $giveup$, which allows a node to terminate broadcast on sensing its neighborhood has received the message. We further study the effect of message segmentation on various types of topologies like d-regular graph, random graph etc. and observe that even for simple $push$ technique, there exists an optimal d for which the dynamics becomes fast. We also simulate our results on real traces and finally provide some suggestions for network designers which we believe will help in faster message dissemination and lesser wastage, especially in case of dynamic networks.**

I. Introduction

The study of broadcast over unstructured and mobile networks always assumes that the size of the message is small enough to be transfered from one node to other on the short durations of contacts between the nodes. Contrary to this, in this paper, we explore the idea of "segmented messages" where we assume that the duration of a contact between the nodes is not always sufficient for the transfer and therefore the message might need to be segmented/divided into sub-parts and sent individually. At the ethernet level such techniques of segmented broadcast are often termed as pipelined broadcast [15], [19]. We systematically study the effect of the size and partition structure of the message on the broadcast time.

In specific, we investigate in details the effect of message segmentation as well as the message transfer protocols on the overall broadcast delay and message wastage. We assume that a big file is split into k (> 1) packets and at the beginning, there is only one sender node in the network that has all the packets. Further, a node can transfer only one packet in a single contact opportunity, and it can do so only when it has received all the packets constituting at least one segment of the message. When all the nodes present in the network have eventually received all the packets, broadcasting is assumed to be complete.

Initially, we investigate the push transfer protocol whereby messages are 'pushed' by the node holding a message to the node not having the message [3], [12]. We attempt to study the effect in different types of topologies e.g., complete graph, d-regular tree, d-regular graph, random graph (with average degree d). For a complete graph topology of size n and the simple push case we show that the overall time required for the broadcast in case of segmented messages scales as $n^{\frac{k-1}{k}}$ where $k > 1$ is the number of packets. This is in sharp contrast to the single message case (k = 1) where it has been shown that broadcasting time scales as $\log(n)$ [2], [9]. Another remarkable observation is that in topologies like d regular tree, d regular graph and random graph, for even the simple push transfer protocol, one can find an optimal value of $d(d > 1)$, for which the broadcast delay and wastage is minimum. (section III). As a corollary, through simulations on real traces, we identify that for two networks with the same number of nodes, broadcast time required is far smaller for the one with lesser edge density. This finding, we believe, indicates a very crucial point – a sparse communication network per se is not disadvantageous.

However, we observe that push transfer protocol results in a large number of useless contacts. An unsuccessful/useless contact here refers to a case where a sender node attempts to send a packet to another node who already has the packet. In order to reduce both broadcast time and wastage we propose a combined strategy whereby the nodes in the system initially push and then switch over to pull after a certain percentage (say x) of the nodes have received the full message. We observe that if x is carefully chosen, both gain in broadcast time and reduction in wastage is achieved. However, to determine that $x\%$ of the nodes have indeed received the message, the system needs to maintain a global information which is not feasible in a distributed setting like this. In order to circumvent this problem, we introduce a distributed version of the previous technique along with "give-up" mechanism whereby nodes attempt to discover their neighborhood by maintaining a history of all previous contacts and stops participating in the broadcast after a certain number of unique unsuccessful contacts. This algorithm when tested over Gnutella topologies is found to yield lesser broadcast delay without significant increase in wastage. We believe, our algorithm with minor modifications is applicable to a wide range of dynamic networks [4], [8], [18].

978-1-4673-7131-5/15/$31.00 ©2015 IEEE

II. Algorithm outline and simulation results

In this section, we describe the overall framework and message transfer techniques that we assume for the rest of our study and also provide the key results.

A. Agent configuration and network setup

We consider a network topology $G =< V, E >$ where each node in V represents an agent of the network and any link in E represents a contact opportunity between a pair of nodes (agents) in the whole time span through which the network is active. So for any node (agent) n_i in this network, its one hop neighbors are the nodes (agents) which are within the connection proximity of n_i and at each time step n_i at random can connect to any one of them.

B. Message configuration

We consider that a message $\mathcal{M}$ is divided into a set of m packets, i.e., $|\mathcal{M}| = m$. Packets in $\mathcal{M}$ are grouped into a set $\mathcal{S}$ of s segments, where each segment consists of $k = m/s$ packets (i.e., s can take only those values for which $m \bmod s = 0$). For instance, if $\mathcal{M}$ constitutes of 4 packets and 2 segments then each of the segment is composed of 2 packets. Note that in this paper we mostly consider that there is only a single segment in the message ($m = k$) unless specified otherwise.

C. Transfer protocol

In this framework, transfer of a message during a contact refers to the transfer of one single packet of any segment of a message. Transfer of a packet from u_i to v_j during a contact can take place only when u_i qualifies as a *sender* by having all the packets of at least one of the message segments. The two basic modes of message transfer that we consider are the push and the (restricted) pull epidemic.

Push technique:

- *Step 1:* At any time step, u_i (already a sender) establishes a communication link with v_j, from its neighborhood and finds an exclusive set of packets that u_i has but v_j does not have in its buffer.
- *Step 2:* If u_i can find such a (non-empty) set, then it transfers only one packet from this set to v_j.

Pull technique:

- *Step 1:* At any time step between v_j and u_i, v_j establishes a communication link with u_i and requests for a packet that it has not received yet.
- *Step 2:* Given that u_i has already become a sender it first finds such a packet from its own buffer and then transfers a copy of it to v_j.

Note that in the traditional pull technique, u_i, being a sender, may transfer more than one packet if it gets multiple simultaneous requests from more than one agent at any particular time step. However, unlike the traditional case, here u_i can serve only one request in case there are multiple pull requests. Such a restriction actually allows for conservation of both battery power and network bandwidth in resource constrained scenarios like DTN. Also note that we consider the duration of each communication link is long enough for the transfer of at least a single packet.

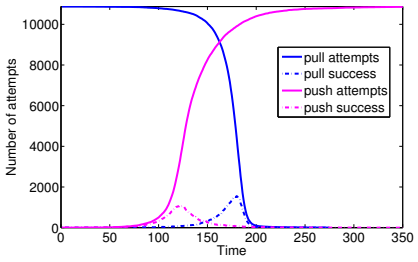

Fig. 1. Pull attempts, successful pulls, push attempts, successful pushes versus time for gnutella1 network

D. Metrics of interest

We are interested to evaluate the performance of a broadcast protocol in terms of two different metrics. The first metric concerns broadcast delay. The second metric centers around a complementary issue of power and bandwidth consumption.

- **Broadcast delay** T^* - this is the time from the point when the message source starts sending the first packet to the point when all the agents in the network have received the entire message. $E(T^*)$ denotes the expected broadcast time. In addition, we are also interested in the time T_i which is the minimum time at which there are i senders (except the source) in the network, and especially in T_1 since, as we shall see, that this is the prime determinant of the entire broadcast time.
- **Broadcast wastage** C_m^* - let C_l and C_p be the total number of communication links that get established in the network and the total number of successful packet transfers respectively. We define the broadcast wastage as $C_m^* = (C_l - C_p)/C_l$. This metric essentially measures the number of useless contacts in the network during which no packet could be transferred and is therefore a direct cause of energy wastage.

E. Broadcast algorithms

We consider following four algorithms for broadcasting messages-

1) Blind push (B-P): An initiator node is the one which has the full message in the beginning. At each time step all the nodes in the system having the full message communicate with a node in their proximity and try to $push$. At the end of each time step all the nodes which have received all the packets qualify as sender in the next time step. The algorithm terminates when all the nodes in the system have the full message.

2) Blind pull: At each time step all the nodes in the system not having the full message, communicate with a node in their proximity and try to $pull$. At the end of each time step the nodes which have received the full message stops pulling from the next time step. The algorithm terminates when all the nodes in the system have the full message.

3) Strategy-x% automatic switch from push to pull: In this Strategy (X-P-P), the nodes in the system follow $Blind-push$ initially and then switch to $Blind-Pull$ once x% of the nodes in the system have the full message. The algorithm terminates when all the nodes in the system have the full message.

We consider the $Blind-push$ and $Blind-pull$ algorithm on a network and check how efficiently they perform over a broadcast time window. In figure 1 we plot the number of attempts and the number of successful ones for the above two cases on gnutella1 network (described later in this section). This clearly shows pull mechanism performs poorly in the beginning but picks up after a certain percentage of nodes have become senders. We observe just the opposite behavior for push mechanism. Our X-P-P strategy is based on this idea to obtain the best out of both the strategies.

The X-P-P strategy cannot be implemented in a practical setting because the nodes in the system need to maintain a global information of the percentage of nodes in the system having the full message which is difficult in a distributed setting like this.

4) A distributed version of X-P-P strategy: Here, we introduce a new strategy $Push-pull-with-giveup$ (P-P-G) which approximately mimics the X-P-P strategy in a distributed setting and it functions in the following way- Initially there is a single node in the system which has the full message. At each time step the sender nodes in the system communicate with one of the nodes in their proximity and try to $push$. Among all the other non-sender nodes in the system those which have at least a single packet (i.e., nodes which have participated in a message transfer at least once and hence are aware of the broadcast) try to pull. Each node also keeps a local history regarding the number of unsuccessful communications it has participated in and once this exceeds a threshold, it 'gives-up' and no longer participates in the broadcast. Once all the nodes have 'given-up', the broadcast terminates.

F. Dynamic topology

We performed our experiments on gnutella snapshots and on synthetic topologies like complete graph, regular tree, regular graph and random graph. A topology specifies the potential neighborhood of a node - a node at each time step connects randomly to one of these nodes. A complete graph topology would indicate that the node can connect to any other node in the network while for other sparser topology it would connect only to a subset of them.

III. Experiment on different network topologies

In this section we systematically study the effect of topology of the underlying contact network on the broadcast time and wastage and come up with some suggestions which we feel will be helpful while designing networks. First we analyze the B-P algorithm on complete graph topology and observe that the broadcast time scales as $n^{\frac{k-1}{k}}$. We then analyze B-P algorithm on other sparser topologies like regular graph, regular tree and random graph. In particular, we wish to check whether the average degree (d) of the underlying contact network influences the performance metrics. In the later part of this section we make a comparative study of different broadcast strategies (discussed in section II) and also reinspect into their sparser variants to identify the effect of lowering the value of d through removal of edges without hampering network connectivity.

A. Blind push on complete graph

We analyze the performance of B-P on complete graph topology. A complete graph topology indicates that a node in the system can communicate with any other node. We provide numeric evidence (with analytical support) that the ratio of the overall broadcast time T^* and the time to create the first sender, i.e., T_1 converges to a constant which only depends on k. For this purpose we plot in figure 2 the values of $Av(T^*)$ and $Av(T_1)$ respectively as we vary n where $Av(y)$ represents the average of the quantity y over several simulation runs. We report this distribution for four different values of m (2, 4, 8 and 16), in each case assuming that there is only one message segment, i.e., $k = m$. Note that for B-P, the two quantities $Av(T^*)$ and $Av(T_1)$ exhibit a very similar profile irrespective of the value of m chosen. In the same figure we also plot the function $n^{\frac{k-1}{k}}$ suitably scaled by a constant to show how the theoretical results which we provide next, closely approximate the numerical simulations.

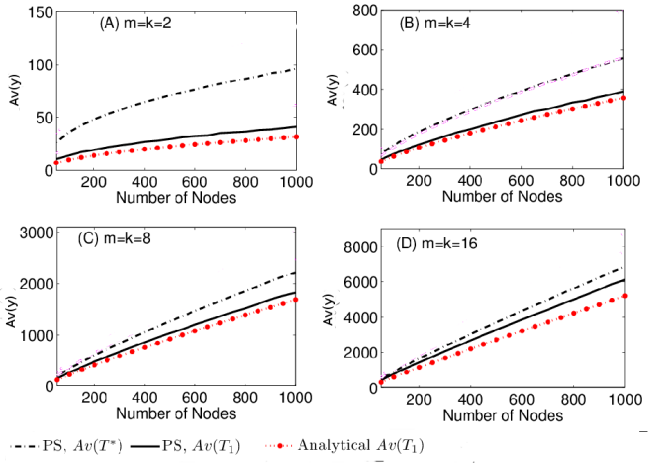

Fig. 2. $Av(T^*)$ and $Av(T_1)$ versus the number of nodes (n). Results are presented for (A) $m = k = 2$, (B) $m = k = 4$, (C) $m = k = 8$, and (D) $m = k = 16$. The plots also contain the suitably scaled function $n^{\frac{k-1}{k}}$ for each case.

1) Analytical estimate: We initially start by considering a message which has $m = 2$ packets and $s = 1$ segment (i.e., the segment has $k = 2$ packets). We compute the expected time to obtain the first sender[1] $E(T_1)$. It is important to note that T_1 is an indicator for the total broadcast time T^* since the remaining growth is of logarithmic order as after creation of the first sender, several senders are produced at regular interval thus speeding up the transfer exponentially. We provide an outline of the analytical expression, the detail is more involved and is ommitted in the interest of space.

At time T_0, an initiator is created and it has the full message. Since we are considering message size of 2 so the first sender can be created at least in 2 time steps which is possible if the same node is selected in these two time steps. Next we calculate $Pr\{T_1 = t\}$ that is the probability that the first sender is created at time t. This implies for the $t-1$ time steps, only the nodes without any packets are selected and at time step t one from the $t-1$ nodes are selected. So we have

[1]Note that first sender refers to the first node (other than the initiator) which receives the full message during the broadcast phase.

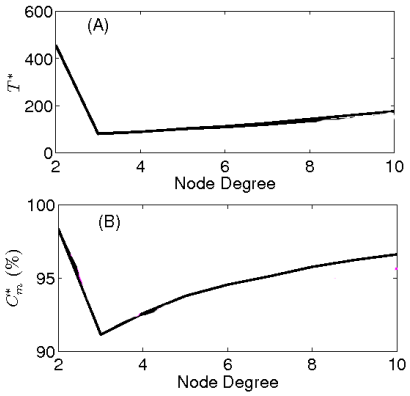

Fig. 3. (A) Broadcast time and (B) broadcast wastage versus different values of d for B-P. The parameters values are $n = 200, m = 4, k = 2$.

$$\Pr\{T_1 = t\} = (1 - \frac{1}{n})(1 - \frac{2}{n})(1 - \frac{3}{n})...(1 - \frac{t-2}{n})(\frac{t-1}{n})$$
$$= \frac{t-1}{n}\prod_{l=1}^{t-2}(1 - \frac{l}{n}), t \geq 2$$

Infact if τ_i represent the time to create the first sender then it can be shown that the recursion $\frac{E(\tau_i)}{E(\tau_{i-1})} = \left(1 - \frac{3}{2i}\right)$ holds. From this one can show that $E(T^*) = 2 * E(T_1)$

Generalizing for case where $k > 2$, we calculate the time to create the first sender T_1. To do it we look into how the number of nodes with exactly l packets (say) grow with t (l varies from 1 to k -1). We can show that the number of nodes with l packets grow as $\frac{t^l}{(l)!n^{l-1}}$

With this estimation we can now proceed in the same way as we did in case of $k = 2$ case and show that

$$\Pr(T_1 = t) \sim \frac{t^{k-1}}{(k-1)!n^{k-1}}\prod_{i}^{t}\left(1 - \frac{i^{k-1}}{(k-1)!n^{k-1}}\right)$$
$$\sim \frac{t^{k-1}}{(k-1)!n^{k-1}}e^{-\frac{t^k}{k!n^{k-1}}}$$

From the above probability distribution of T_1 we calculate $E(T_1) = \sum t * \frac{t^{k-1}}{(k-1)!n^{k-1}}e^{-\frac{t^k}{k!n^{k-1}}}$ and through a lengthy calculation can show that $E(T_1)$ is of the order $n^{\frac{k-1}{k}}$.

B. Blind push on sparser topologies

Next we empirically analyze the B-P algorithm on sparser topologies like regular-tree, regular-graph and random-graph. For each topology we consider $n = 200$, $m = 4$ and $k = 2$. Note that here we consider that the message has 2 segments and each segment has $k = 2$ packets. We then vary the average degree d for each of these networks and check how broadcast delay and wastage depend on it. Remarkably, for each of these topologies - regular tree (figure 3), regular graph (figure 4) and random graph (figure 5), one can observe that there is a critical value of d for which we obtain minimum broadcast delay and wastage.

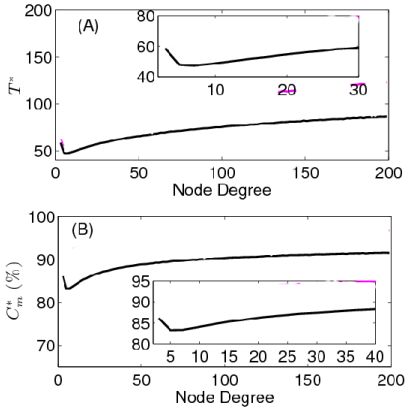

Fig. 4. (A) Broadcast time and (B) broadcast wastage versus different values of d for B-P technique. The parameters values are $n = 200, m = 4, k = 2$. The inset in both the figures show the metrics of interest for the first few values of d to indicate the critical d more appropriately.

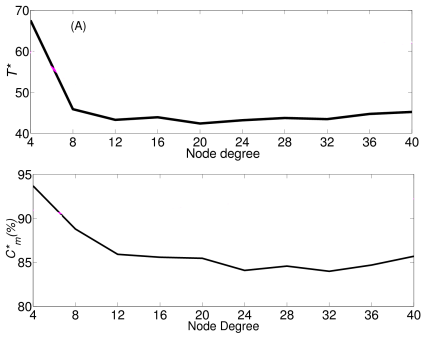

Fig. 5. (A) Broadcast time and (B) broadcast wastage versus average degree for B-P . The parameters values are $n = 200, m = 4, k = 2$.

C. Comparison of different broadcast strategies on Gnutella Topology

We measure the performance of the algorithms (B-P, X-P-P and P-P-G) on three real network traces based on broadcast time, wastage and coverage. The data sets are Gnutella network snapshots namely p2p-Gnutella04 (gnutella1), p2p-Gnutella06 (gnutella2) and p2p-Gnutella25 (gnutella3) [10], [17] taken on dates August 4, August 6 and August 25, 2002 respectively. The gnutella1 network has 10876 nodes and 39994 edges in its largest connected component. Corresponding number of nodes and edges in the largest connected component in gnutella2 and gnutella3 are respectively 8717, 31525 and 22663, 54693.

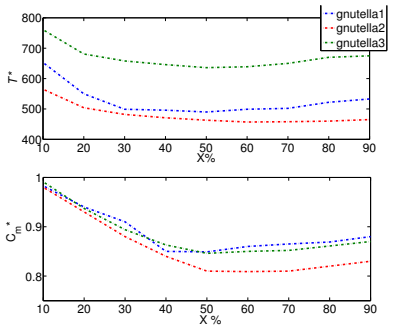

Fig. 6. Average broadcast time and wastage versus x for gnutella1,gnutella2 and gnutella3

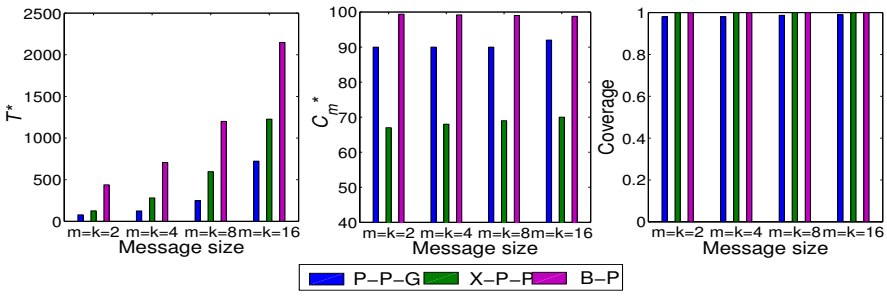

Fig. 7. (A) Broadcast time versus message size (B) Wastage versus message size (C) Coverage versus message size for gnutella3

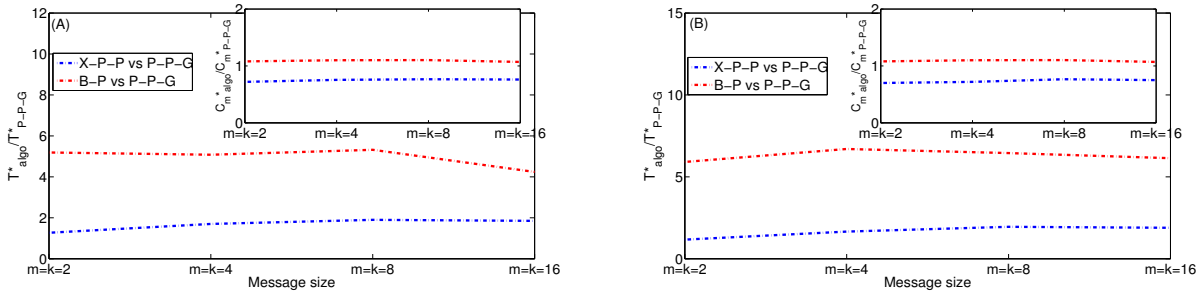

Fig. 8. Gain in broadcast time of P-P-G over X-P-P and B-P [Inset shows gain in wastage] for (A)gnutella1 and (B)gnutella2 networks. Note: algo = B-P/X-P-P

We simulate these algorithms for varying message sizes. We perform our simulations on peer-to-peer systems specifically as our study can find a major application in such systems.

For simulating the X-P-P algorithm in particular we first need to obtain the best value of x for each network and then run the simulations for varying message sizes. In figure 6, we show how the broadcast time and wastage varies with x for networks gnutella1, gnutella2 and gnutella3. We observe that the best value of x lies around 50% for the gnutella1 network. Similarly the obtained value of x are found it to be around 50% and 60% for gnutella2 and gnutella3 respectively. In figure 7 we plot the broadcast time, wastage and coverage for gnutella3 network. We observe that with P-P-G we gain in both broadcast time and wastage with respect to B-P. With respect to X-P-P, P-P-G offers better broadcast time but with a higher wastage. For gnutella1 and gnutella2 networks we plot (figure 8) the ratio of broadcast time and wastage of X-P-P and B-P over P-P-G for different message sizes. We observe that across different message sizes, on an average the gain in broadcast time of X-P-P over P-P-G and B-P over P-P-G are 5.45 and 1.5 respectively for gnutella1 network while for gnutella2 network the corresponding values are 5.9 and 1.6 respectively. Corresponding values for wastage are 0.75 and 1.10 respectively for gnutella1 network and 0.72 and 1.06 respectively for gnutella2 network. So with P-P-G we gain in both broadcast time and wastage with respect to B-P while with respect to X-P-P we gain in broadcast time without significant increase in wastage. Actually, X-P-P provides the best optimization between broadcast time and wastage but it would be hard to implement in a distributed setting. Our proposed algorithm (P-P-G) presents the best possible trade-off of delay and wastage guaranteeing almost 100% coverage and can be implemented in a distributed fashion with almost negligible computational overhead.

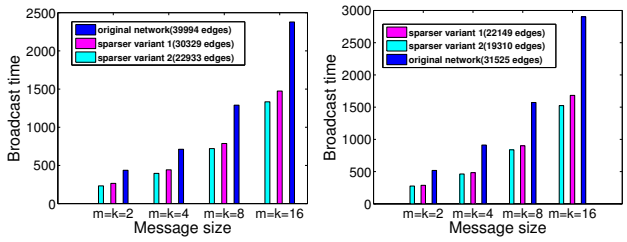

Fig. 9. Broadcast time for the gnutella snapshots and their sparser variants versus different values of message sizes for (a). gnutella1 and (b). gnutella2

1) Effect of Degree on Broadcast time: In earlier part of this section we observed that irrespective of the topology one is able to a obtain critical value of d for which the broadcast time is minimum. So we performed simulations on sparser variants of these gnutella networks and observed that broadcast time reduces even for the B-P. To obtain sparser variants, we considered each gnutella snapshot and randomly removed some of the edges without hampering the network connectivity. From figure 9 we observe that the broadcast time reduces significantly in case of the sparser variants in comparison to the original network. Hence, while designing a network it is advisable to keep the network sparse rather than creating unnecessary connections between the nodes. This, as our results indicate, should lead to a faster dissemination of messages.

IV. Related Works

Efficient data dissemination in dynamic and distributed networks like delay-tolerant, peer-to-peer, ad-hoc networks is useful for many applications and the existing studies have shown combining the push and the pull epidemic protocols can be an efficient approach because of their inherent robustness.

In an interesting study in [6], it has been shown that the cost (in terms of time and resource utilization) to reach the last 10% of the agents is much higher compared to the cost of reaching the first 90% of the agents during broadcasting in DTNs. To solve this problem, the authors in this study propose a controlled broadcasting technique where the broadcast start using push technique, and towards the end of broadcast the agents switch to adopt the pull technique (instead of push). The study in [11] has proposed a cooperative file sharing mechanism in a practical DTN framework, where only a few agents have Internet connectivity. The authors show that, to efficiently spread a file in the network, the agents that are actually downloading the files from Internet or cellular networks can use the pull technique, and then they later on can push the received message to other agents. The study in [13] has proposed a framework for DTNs, where the agents in the network form communities located in different geographical regions, and the agents in different communities use different network technology such as Bluetooth, WiMAX, etc. Agents located within the same region may use the pull technique for information sharing, whereas to spread it across the different regions they can use the push technique.

In addition, there have also been few works on fragmentation of larger sized messages to make them suitable for transmission in DTNs and Peer-to-peer networks. In [16] the authors introduce various strategies of message fragmentation independent of routing algorithm and evaluate their impact in DTNs. Some of the more recent works include [1], [7]. In peer-to-peer systems there are also numerous instances where the authors have tried to efficiently combine the push and the pull epidemic protocols. In [18] the authors propose a combined strategy by interleaving between push and pull and show that k pieces can be disseminated from a single source to n users in $9(k+\log n)$ time. [5] introduces a system called Pulp which proposes a data dissemination approach by limiting push and also reducing the redundant pulls. Some other data dissemination strategies combining push and pull have been proposed in [12], [14]. Since peer-to-peer systems deal with dissemination of large files fragmenting them before spreading is an obvious choice as is considered in all the above works.

In this paper, we study for the first time, the systematic coupling of two different issues that have been dealt in the literature only independently and in parts. These two issues concern (a) the effect of message segmentation and (b) the technique adopted for transferring the message, especially the give-up technique allowing for a deterministic termination in a completely distributed fashion on broadcast time and wastage.

V. Conclusion

The significance of the paper lies in defining a new problem in the space of information dissemination and broadcast in unstructured dynamic networks. In this paper we show through simulation and initial analytical results that the speed at which segmented data can be disseminated over dynamic network is different from it comprising a single segment. We explore the impact of the problem in different topologies and surprisingly notice that mere dense topology is not of much help in a dynamic setting. We also try to propose a practical strategy which can optimize the dual (conflicting) objective of speed and wastage. We believe these initial results can be enriched by tackling the problem in a more practical setting like allowing more than one message transfer in one time step, considering the order of the message segments, considering wider variants of topology snapshot etc. This along with more rigorous theoretical analysis would be our immediate future work.

References

[1] Ahmed B Altamimi. *On Message Fragmentation, Coding and Social Networking in Intermittently Connected Networks*. PhD thesis, University of Victoria, 2014.

[2] Flavio Chierichetti, Silvio Lattanzi, and Alessandro Panconesi. Rumour spreading and graph conductance. In *SODA*, pages 1657–1663, 2010.

[3] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *ACM Symposium on Principles of distributed computing*, pages 1–12. ACM, 1987.

[4] Patrick Th Eugster, Pascal A Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys (CSUR)*, 35(2):114–131, 2003.

[5] Pascal Felber, Anne-Marie Kermarrec, Lorenzo Leonini, Etienne Rivière, and Spyros Voulgaris. Pulp: An adaptive gossip-based dissemination protocol for multi-source message streams. *Peer-to-Peer Networking and Applications*, 5(1):74–91, 2012.

[6] Gian Paolo Rossi Francesco Giudici, Elena Pagani. Self-adaptive and stateless broadcast in delay and disruption tolerant networks. Technical report, Universit a degli Studi di Milano, 2008.

[7] Philip Ginzboorg, Valtteri Niemi, and Jörg Ott. Message fragmentation for a chain of disrupted links. *Computer Communications*, 48:84–97, 2014.

[8] Zhigang Jin, Jia Wang, Sainan Zhang, and Yantai Shu. Epidemic-based controlled flooding and adaptive multicast for delay tolerant networks. In *Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing (UIC/ATC), 2010 7th International Conference on*, pages 191–194. IEEE, 2010.

[9] Stefano Leonardi, Alessandro Panconesi, Paolo Ferragina, and Aristides Gionis, editors. *Rumor Spreading in Random Evolving Graphs*. ACM, February 2013.

[10] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

[11] Cong Liu, Jie Wu, Xin Guan, and Li Chen. Cooperative file sharing in hybrid delay tolerant networks. ICDCSW '11, pages 339–344, Washington, DC, USA, 2011. IEEE Computer Society.

[12] R Lo Cigno, Alessandro Russo, and Damiano Carra. On some fundamental properties of p2p push/pull protocols. In *Communications and Electronics, 2008. ICCE 2008. Second International Conference on*, pages 67–73. IEEE, 2008.

[13] Mirco Musolesi and Cecilia Mascolo. A framework for multi-region delay tolerant networking. In *ACM workshop on Wireless networks and systems for developing regions*, WiNS-DR '08, pages 37–42, New York, NY, USA, 2008. ACM.

[14] Oznur Ozkasap, Mine Caglar, and Ali Alagoz. Principles and performance analysis of second: A system for epidemic peer-to-peer content distribution. *Journal of Network and Computer Applications*, 32(3):666–683, 2009.

[15] Pitch Patarasuk, Xin Yuan, and Ahmad Faraj. Techniques for pipelined broadcast on ethernet switched clusters. *Journal of Parallel and Distributed Computing*, 68(6):809–824, 2008.

[16] Mikko Pitkänen, Ari Keränen, and Jörg Ott. Message fragmentation in opportunistic dtns. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–7, Washington, DC, USA, 2008. IEEE Computer Society.

[17] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *arXiv preprint cs/0209028*, 2002.

[18] Sujay Sanghavi, Bruce Hajek, and Laurent Massoulié. Gossiping with multiple messages. *Information Theory, IEEE Transactions on*, 53(12):4640–4654, 2007.

[19] Jerrell Watts and Robert Van De Geijn. A pipelined broadcast for multidimensional meshes. *Parallel Processing Letters*, 5(02):281–292, 1995.