GADGET SVM: a Gossip-bAseD sub-GradiEnT SVM solver

Chase Hensel CMH2135@COLUMBIA.EDU
Department of Computer Science, Columbia University New York, NY 10027

Haimonti Dutta HAIMONTI@CCLS.COLUMBIA.EDU
Center for Computational Learning Systems, Columbia University New York, NY 10025

Keywords: Distributed Data Mining, Support Vector Machines, Distributed Optimization

1. Introduction

Distributed environments such as federated databases, wireless and sensor networks, Peer-to-Peer (P2P) networks are becoming increasingly popular and well-suited for machine learning since they can store large quantities of data on a network. The distributed setting is complex in part because network topologies are often dynamic and data available to algorithms changes frequently. Furthermore, in many distributed scenarios (such as sensor networks) nodes may have limited resources. Distributed Data Mining (DDM, (Kargupta & Chan, 2000), (Demers et al., 2002), (Guo & (editors), 1999), (Provost, 2000)) and Machine Learning algorithms created for these settings must have high utility, use little communication cost, work on dynamic networks and be computationally efficient.

Our goal is to create a distributed decentralized support vector machine (Vapnik, 1998) solver which achieves the above goals. As most Distributed SVM (DSVM) variants either necessitate some sort of centralized server (Kokiopoulou & Frossard, 2006) or work on specific network topologies (Lu et al., 2008) there is a need for such a DSVM algorithm. In the distributed classification setting, nodes in a network have partial sets of training examples and must work together to learn a classifier. Formally, given a network, G of m nodes, each with its own set of training examples, $S_i = \{(\mathbf{x_j}, y_j)\}_{j=1}^{n_i}$, where n_i is the number of examples at node i, $\mathbf{x_j} \in \mathbb{R}^d$, and $y_j \in \{+1, -1\}$, our goal is to find the global minimizer to the problem

$$\min_{\mathbf{w}} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{n} \sum_{i=1}^{m} \sum_{j=1}^{n_i} l(\mathbf{w}; (\mathbf{x_j}, y_j)), \qquad (1)$$

where l is the loss function defined as $l(\mathbf{w}; (\mathbf{x}, y)) = \max\{0, 1 - y\langle \mathbf{w}, \mathbf{x} \rangle\}$, λ is the SVM regularization parameter, and n is the total number of examples in the network. Note that this version of SVM differs from the standard formulation in two ways – it contains a double sum because the loss is aggregated across a network and the loss function does not include a bias parameter b.

In this paper, we describe and analyze Gadget, a simple iterative algorithm for computing an ϵ-accurate solution to Eq. 1. Our algorithm runs in $\tilde{O}\left(\frac{d(n_i' + \deg' \tau_{mix})}{\lambda \epsilon}\right)$ time, and requires each node to send $\tilde{O}\left(\frac{d \deg' \tau_{mix}}{\lambda \epsilon}\right)$ messages, where n_i' is the maximum number of examples at any node, d is the total number of non-zero features, τ_{mix} denotes the mixing time of a random walk on network G, and $\deg'$ is the maximum degree of any node. The Gadget algorithm is a distributed variant of the sub-gradient projection method introduced in (Shalev-Shwartz et al., 2007). Due to the additional complication of the distributed setting, our method does not use true sub-gradient updates and instead uses approximate sub-gradient updates obtained via an average network consensus protocol, Push-Sum, found in (Kempe et al., 2003; Kempe & McSherry, 2004).

Before describing Gadget, we would like to frame our efforts in the context of some of the recent work on distributed SVM, and other sub-gradient SVM algorithms.

Distributed Support Vector Machines: (Syed et al., 1999) proposed a DSVM algorithm which found support vectors locally at each node and then sent them to a central server for processing. This approach was promising because it was both highly parallel, and worked on arbitrary topologies; however, it did not find the global optimal solution and its communication cost depended on the total size of its data-set. The algorithm was improved in (Caragea et al., 2005)

by allowing the centralized server to send the support vectors back to the distributed nodes and then repeating the process until a global optimum was achieved. Despite reaching optimality, this approach was slow due to extensive communication costs. Another approach, Cascade SVM (Graf et al., 2005) worked on a top-down network topology and quickly generated a globally optimal solution. In a similar vein to the Cascade SVM, (Lu et al., 2008) created a DSVM variant suited for Kurtowski graphs. Both algorithms required specific network topologies and the transfer of support vectors between nodes resulting in large communication complexity. Gadget works on arbitrary networks with a communication complexity independent of the data-set size, and only linear dependence on network size. Furthermore, Gadget always works towards optimality and can produce an ϵ-accurate solution for any $\epsilon > 0$.

Gradient descent SVM algorithms: Gradient based optimization methods exhibit slow convergence rates; however, the computational demands of large data-sets have caused descent methods to remain popular despite faster converging cousins based on Newton's method. Shalev-Shwartz (Shalev-Shwartz et al., 2007), introduced a stochastic projected sub-gradient algorithm called Pegasos which is the immediate precursor of Gadget. However, Pegasos requires the use of exact sub-gradient updates, whereas this requirement is impossible on a distributed network. Consequently, Gadget must settle for *approximate* sub-gradient updates during each iteration.

2. Gadget Algorithm

In this section we will introduce and describe the Gadget algorithm for solving Eq. 1. The algorithm

Table 1. Summary of Notation

$L_g(w)$	Global hinge loss of vector w
$\hat{L}_{g,i}^{(t)}$	Node i's approximate global loss at time t
$\tilde{L}_g^{(t)}$	Network average approximate loss at time t
$\hat{w}_i^{(t)}$	Node i's weight vector at time t
$w_i^{(t+\frac{1}{2})}$	Node i's update at time t
$\hat{w}_i^{(t+\frac{1}{2})}$	Node i's approximate network average update
$\bar{w}_i$	Sum of first t local weight vectors
$\tilde{w}^{(t)}$	Network average weight vector
$\alpha^{(t)}$	Learning parameter
γ	Push-Sum error term

Algorithm 1 Gadget(λ, S_i, T, B)

$\hat{\mathbf{w}}_\mathbf{i}^{(1)} = \bar{\mathbf{w}}_\mathbf{i} = 0$
for $t = 1, \ldots, T$ **do**
  Set $S_i^+ = \left\{(\mathbf{x}, y) \in S_i : y\langle \hat{\mathbf{w}}_\mathbf{i}^{(t)}, \mathbf{x}\rangle < 1\right\}$
  Set $L_i^{(t)} = \sum_{(\mathbf{x},y)\in S_i^+} y\mathbf{x}$
  Set $\hat{L}_{g,i}^{(t)} \leftarrow$ Push-Sum$(B, L_i^{(t)}, n_i)$
  Set $\alpha^{(t)} = \frac{1}{\lambda t}$
  Set $\mathbf{w_i}^{(t+\frac{1}{2})} = (1 - \lambda\alpha^{(t)})\hat{\mathbf{w}}_\mathbf{i}^{(t)} + \alpha^{(t)}\hat{L}_{g,i}^{(t)}$
  Set $\hat{\mathbf{w}}_\mathbf{i}^{(t+\frac{1}{2})} \leftarrow$ Push-Sum$(B, \mathbf{w_i}^{(t+\frac{1}{2})}, 1)$
  Set $\hat{\mathbf{w}}_\mathbf{i}^{(t+1)} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\hat{\mathbf{w}}_\mathbf{i}^{(t+\frac{1}{2})}\|}\right\}\hat{\mathbf{w}}_\mathbf{i}^{(t+\frac{1}{2})}$
  $\bar{\mathbf{w}}_\mathbf{i} = \bar{\mathbf{w}}_\mathbf{i} + \hat{\mathbf{w}}_\mathbf{i}^{(t+1)}$
  $\bar{\mathbf{w}}_\mathbf{i}/t$ is current estimate of the maximum margin hyperplane
end for

receives as input two parameters: T - the number of iterations to perform, and λ. In general we assume that each node in the network is aware of not only their local information, but also the current iteration and part of a stochastic matrix B used during network communication.

Gadget begins by flooding the network with the SVM regularization parameter λ, and T. Each node then initializes $\hat{\mathbf{w}}_\mathbf{i}^{(1)}$ and the current total $\bar{\mathbf{w}}_\mathbf{i}$ (described in Table 1) which is later used to generate the output. During each iteration, the nodes flood the network to tell each other t. Each node calculates its local SVM loss, and then uses Push-Sum (Algorithm 2) to compute the approximate network-wide SVM loss.

From Push-Sum, node i obtains an approximate SVM loss $\hat{L}_{g,i}^{(t)}$ which has at most some known relative error, γ. Node i then changes the hyperplane via the sub-gradient update rule.

$$\mathbf{w_i}^{(t+\frac{1}{2})} = (1 - \lambda\alpha^{(t)})\hat{\mathbf{w}}_\mathbf{i}^{(t)} + \alpha^{(t)}\hat{L}_{g,i}^{(t)} \qquad (2)$$

The second call to Push-Sum ensures that each node has a value of $\mathbf{w_i}^{(t+\frac{1}{2})}$ with relative error at most γ, and thus maintains overall numeric stability. In the same spirit as (Shalev-Shwartz et al., 2007), the algorithm next projects $\hat{\mathbf{w}}_\mathbf{i}^{(t+\frac{1}{2})}$ onto the ball of radius $1/\sqrt{\lambda}$ in order to bound the maximum sub-gradient. A specific discussion on γ is deferred to the analysis section.

As described in (Kempe & McSherry, 2004), the Push-Sum protocol deterministically simulates a random

Algorithm 2 Push-Sum$(B, (V), w)$

Each node starts with w_i equal to input weight.
All nodes set S_i equal to local input vector V
loop
  Set $S_i = \sum_{j \in N(i)} b_{j,i} S_j$
  Set $w_i = \sum_{j \in N(i)} b_{j,i} w_j$
end loop
Return $\frac{S_i}{w_i}$.

walk across G and estimates network sums. If we create an arbitrary stochastic matrix $B = (b_{i,j})$ across our network such that if there is no edge from i to j in G, then $b_{i,j} = 0$, and otherwise B is ergodic and reversible, then Push-Sum converges to a γ-relative error solution in $O(\tau_{mix} \log \frac{1}{\gamma})$, where τ_{mix} is the mixing speed of the Markov Chain defined by B. Informally, τ_{mix} is the the time until B is "close" to its steady state distribution. An obvious choice for B is to use the random walk on the underlying topology, i.e. $b_{i,j} = \frac{1}{\deg i}$. In general, we don't expect the nodes to know τ_{mix}, and(Kempe & McSherry, 2004) describe a simple technique with multiplicative overhead for the nodes to calculate a stopping time for Push-Sum provided they have an upper bound on network diameter.

We should note that the use of an approximate consensus protocol like Push-Sum is necessary because it is impossible to obtain a 100% accurate aggregate sum without complete knowledge of the network. Providing nodes with this information is problematic because it is liable to change during operation, and requires additional communication than otherwise necessary.

3. Analysis

Before analyzing the Gadget algorithm, we need to introduce strong convexity and the notion of a subdifferential. After introducing these two items, we will obtain the convergence rate of Gadget in terms of the convergence of the projected sub-gradient method, and the network error. We will prove a lemma about the SVM loss function and then apply the relative error bounds of Push-Sum to obtain the true convergence rate.

Informally, a strong convex function is a function who's gradient is always changing, or equivalently who's hessian is always positive definite. Unfortunately, the SVM objective function is not differentiable and our analysis must rely on the following, more general, definition of strong convexity using subdifferentials. The *sub-differential* of $f(x)$, denoted $\partial f(x)$, is the set of all tangent lines which can be drawn under $f(x)$. Moreover, each vector $v \in \partial f(x)$ is called a *sub-gradient* of $f(x)$. The following is the formal definition of strong convexity.

Definition 1 *A function $f : \mathbb{R}^d \to \mathbb{R}$ is λ-strongly convex if $\forall x, y \in \mathbb{R}^d$ and $\forall g \in \partial f(x)$,*

$$f(y) \geq f(x) + \langle g, y - x \rangle + \frac{\lambda}{2} \|x - y\|_2^2 . \qquad (3)$$

Notice that in the subsequent analysis, R refers to the maximum norm of a feature vector x_i, $c = (\sqrt{\lambda} + R)$ refers to the maximum norm of any sub-gradient of Eq. 1, and furthermore, proofs are omitted due to a lack of space.

Theorem 1 *If $f(w)$ is the function from Eq. 1 and $w^\star$ is its optimal solution then for all $T \geq 1$*

$$\sum_{t=1}^{T} f(\hat{\mathbf{w}}_\mathbf{i}^{(t)}) - f(\mathbf{w}^\star) \leq \frac{c^2}{2\lambda}(\log(T) + 1) + c \sum_{t=1}^{T} \left\| \hat{\mathbf{w}}_\mathbf{i}^{(t)} - \tilde{\mathbf{w}}^{(t)} \right\|_2 + 2\sqrt{\lambda} \sum_{t=1}^{T} \left\| L_g(\tilde{w}^{(t)}) - \tilde{L}_g^{(t)} \right\|_2 .$$

Thm. 1 provides a convergence rate which is identical to (Shalev-Shwartz et al., 2007), except for the two additional error terms due to network communication. The first of these two terms is directly dependent on the relative error of the normalization step; however, a bound on the second term is slightly more involved and requires the following lemma.

Lemma 1 *If $L_g(w)$ is the global loss of the vector w then $\|L_g(w_1) - L_g(w_2)\|_2 \leq R \|w_1 - w_2\|_2$.*

The above lemma bounds the difference between the global loss of two related vectors. The following theorem applies this lemma and Push-Sum relative error to obtain a convergence rate for fixed topology networks.

Theorem 2 *If the conditions of Theorem 1 hold and Push-Sum provides solutions with γ relative error then for all $T \geq 1$*

$$f(\bar{w}) - f(w^\star) \leq \frac{c^2}{2T\lambda}(\log(T) + 1) + 5c\gamma(\lambda + \sqrt{\lambda}).$$

Setting $\gamma = \frac{\epsilon}{10c(\lambda + \sqrt{\lambda})}$, and observing that Push-Sum requires $O(\tau_{mix} \log \frac{1}{\gamma})$ iterations to achieve

relative error γ, we obtain our message complexity. Moreover, observing that the running time of one iteration of Gadget is at most $d(n_i' + \deg' \tau_{mix} \log \frac{1}{\gamma})$, we achieve our asymptotic runtime.

Theorem 2 no longer holds if the aggregate data-set across the network changes during during computation. In particular this implies that Gadget does not have a convergence guarantee when nodes enter and leave the network during runtime. However, we should note that the sub-gradient steps are always taken in the direction of the current global optimal solution, so the algorithm always works towards the true optimum. In future work we will attempt to address this issue and provide convergence bounds in the presence of a changing aggregate data-set.

4. Experiments

In this section we present experimental results which highlight the usefulness of the Gadget algorithm. We first demonstrate its scalability by showing that for various network sizes, Gadget runs within a constant factor over-head of the deterministic variant of the state-of-the-art Pegasos algorithm (Shalev-Shwartz et al., 2007). We next compare the number of messages sent by Gadget to the number required to centralize the distributed data-set for various network sizes. Finally, fixing network size, we show the effect of different values of ϵ on both running time, and message complexity.

Our implementation of Gadget uses a wheel graph for its underlying topology. We made this choice because the mixing time for a random walk on the wheel is essentially a constant (it was less than three for every network size tested). For our experiments, we used the Poker Hand Data Set from the UCI Machine Learning Repository (Asuncion & Newman, 2007). This data-set contains 1,025,010 10-dimensional examples. Five dimensions correspond to the suits of the five cards in a poker hand, and the other five correspond to the rank of each card. Our task was to determine if an example hand contained at least a pair, or if the example hand was only card high. Breaking up the data into 10,000 example chunks, we distributed it to nodes on a simulated network.

Figure 1 a) shows the ratio of Gadget cpu-time to Pegasos cpu-time on the centralized data. Figure 1 b) shows the ratio of the number of messages used by Gadget to the number of messages needed to centralize the data-set. We ran both algorithms with $\lambda = .1$, until we obtained a $\epsilon = .1$ close approximation to the optimal hyper-plane. We implemented both Gadget and Pegasos in Java and ran all experiments on a 2Ghz Intel Core Duo Macbook with 2GB of ram.

The fact that Gadget's overhead decreases with network size is promising because it suggests improved performance with scalability. Moreover, our synthetic network only takes advantage of a 2-times parallel speed up over Pegasos (each node is contained in a different Thread, and our tests were run on a 2-core processor). On a real network, with reasonably spread data, one would expect this speed-up to be proportional to the number of nodes, rather than the constant 2. This implies that Gadget can run faster on large networks than Pegasos can on the aggregate data-set in a central location.

Interestingly, the fractional message savings seems to level off to a constant as network size increases. This suggests that for our choice of topology, the quantity of messages passed increases linearly with the number of nodes in the network. As Gadget uses a fixed number of messages for a given topology, ϵ and λ, fixing these three parameters, message savings is linearly inversely-proportional to the total number of examples. Together, these observations suggest that Gadget is highly suited for large networks with large quantities of examples per node.

In the second experiment, we set the number of rounds per call to Push-Sum to $4 \cdot \tau_{mix} \approx 12$, thereby setting $\gamma = .0001$ (much smaller than necessary), and solved the SVM problem for ϵ ranging from .1 to .01. In doing so, we demonstrate Gadgets performance as accuracy increases. Figure 2 a) shows cpu-time as a function of ϵ, and Figure 2 b) shows message complexity as a function of ϵ. As is expected from our convergence bounds, both cpu-time and message complexity are linearly inversely-proportional to ϵ.

5. Conclusion and Future Work

Network based gradient descent is promising because convergence is proportional to accuracy rather than data-set size. Algorithms like Gadget are extremely scalable because their runtime has only a linear dependence on network size and a linear dependance on the largest data-set at any particular node. Moreover, the use of robust gossip protocols such as Push-Sum in conjunction with robust descent methods such as projected sub-gradient descent is interesting because it enables robust network optimization. While Gadget works on random topologies, it does not yet handle *dynamic* networks, or those where edges are allowed to change and nodes are allowed to enter and leave. This is a key problem for future work.

Also key to the success of DSVM is the incorporation of a bias parameter in the SVM loss function and an investigation into distributed Mercer kernels. As mentioned in (Shalev-Shwartz et al., 2007), the simple addition of a bias parameter is problematic because it removes the strong convexity properties of the SVM objective function.

Mercer kernels are problematic for DSVM because they require an inner product of examples. In the localized setting this merely requires the taking the inner product of each feature with every other feature; however in the distributed setting this requires the computation of an inner product of every example at every node with every example at every other node, or the effective communication of the entire data-set across the network. Inherent to the success of any distributed Mercer kernel will be the removal of this network wide communication.

References

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Caragea, C., Caragea, D., & Honavar, V. (2005). Learning support vector machines from distributed data sources. *AAAI* (pp. 1602–1603).

Demers, A., Gehrke, J. E., & Riedewald, M. (2002). Research Issues in Distributed Mining and Monitoring. *Proceedings of the National Science Foundation Workshop on Next Generation Data Mining (NGDM 2002)*. Baltimore, MD.

Graf, H. P., Cosatto, E., Bottou, L., Dourdanovic, I., & Vapnik, V. (2005). Parallel support vector machines: The cascade svm. In L. K. Saul, Y. Weiss and L. Bottou (Eds.), *Advances in neural information processing systems 17*, 521–528. Cambridge, MA: MIT Press.

Guo, Y., & (editors), R. G. (1999). *High performance data mining: Scaling algorithms, applications and systems*. Kluwer Academic Publishers.

Kargupta, H., & Chan, P. (2000). *Advances in Distributed and Parallel Knowledge Discovery*. AAAI/MIT Press.

Kempe, D., Dobra, A., & Gehrke, J. (2003). Gossip-based computation of aggregate information. *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science* (p. 482). Washington, DC, USA: IEEE Computer Society.

Kempe, D., & McSherry, F. (2004). A decentralized algorithm for spectral analysis. *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing* (pp. 561–568). New York, NY, USA: ACM.

Kokiopoulou, E., & Frossard, P. (2006). Distributed SVM applied to image classification. *IEEE, Int. Conf. on Multimedia and Expo*.

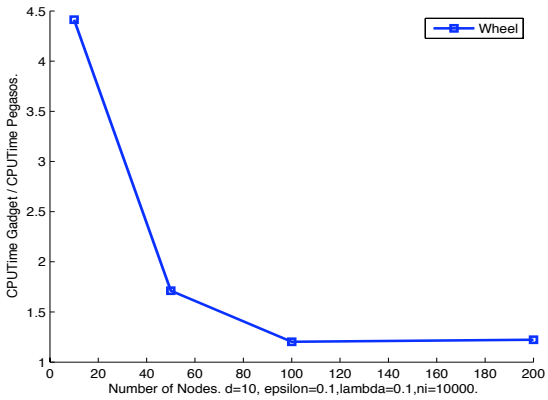
Lu, Y., Roychowdhury, V., & Vandenberghe, L. (2008). Distributed parallel support vector machines in strongly connected networks. *Neural Networks, IEEE Transactions on*, *19*, 1167–1178.

Provost, F. (2000). Distributed Data Mining: Scaling Up and Beyond. In H. Kargupta and P. Chan (Eds.), *Advances in Distributed Data Mining*. MIT/AAAI Press.

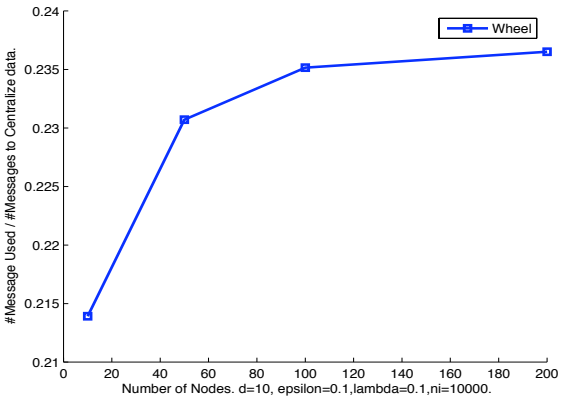
Shalev-Shwartz, S., Singer, Y., & Srebro, N. (2007). Pegasos: Primal estimated sub-gradient solver for svm. *ICML '07: Proceedings of the 24th international conference on Machine learning* (pp. 807–814). New York, NY, USA: ACM.

Syed, N. A., Liu, H., & Sung, K. K. (1999). Handling concept drifts in incremental learning with support vector machines. *KDD '99: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 317–321). New York, NY, USA: ACM.

Vapnik, V. N. (1998). *Statistical learning theory*. Wiley-Interscience.

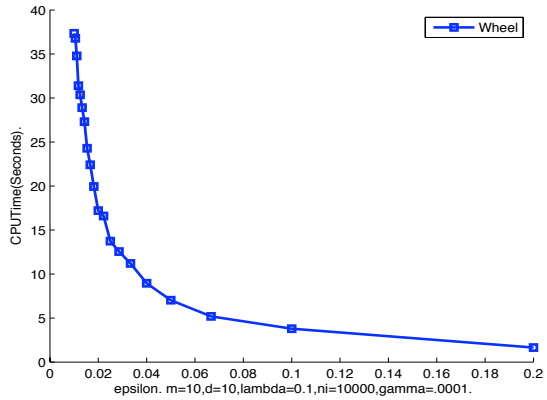


(a) Gadget cpu-time divided by Pegasos cpu-time for various network sizes.

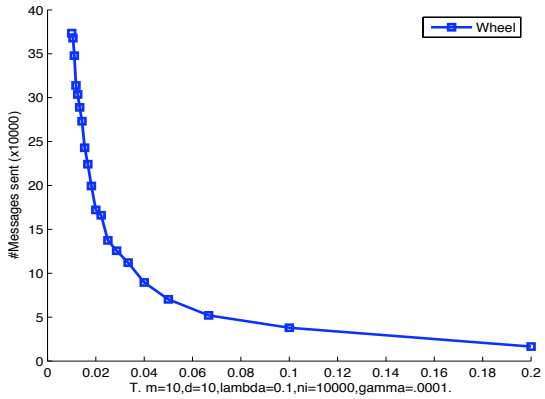


(b) Messages sent by Gadget divided by the number of messages needed to centralize the data under the same conditions as Figure 1(a).

Figure 1. CPU Time and Number of Messages sent by GADGET.



(a) Gadget cpu-time as a function of T for 10-node wheel with fixed gamma.



(b) Number of messages sent by Gadget as a function of T under the same conditions as Figure 2(a).

Figure 2. CPU Time and Number of Messages sent by GADGET for a 10-node wheel.