

On optimal cooperative route caching in large, memory-limited wireless ad hoc networks

Theodoros Salonidis[†] and Leandros Tassioulas^{†,‡}

[†] Department of Electrical and Computer Engineering and Institute of Systems Research
University of Maryland, College Park, MD, USA

[‡] Department of Computer and Communication Engineering
University of Thessaly, Volos, Greece
thsalon@eng.umd.edu, leandros@uth.gr

Abstract—Caching is a popular mechanism for enhancing performance in various layers and applications of computer networking. We introduce both a model and algorithms for caching routing information in large, memory-limited wireless ad hoc networks. Each host can cache only a small fraction of the network and must rely on flooding to acquire information that has not been locally cached. To constrain flooding, the network uses a cooperative caching model where every node provides its route cache contents to others when they flood. Given the host memory capacity limitations, we are faced with the problem of allocating destinations to caches in an efficient manner. We propose the class of Best State/Best Cost (BSBC) cooperative caching algorithms that aim to minimize the overall network search effort.

I. INTRODUCTION

We consider a static wireless ad hoc network with a large number of memory-limited nodes. A representative application is a sheer number of sensors deployed in an area to perform various sensing and communication tasks. Due to memory limitation, nodes cannot maintain routing information for the entire network and use an on-demand ad hoc routing protocol [1][2][3]. According to such a protocol, a node caches routing information only for a subset of the intended destinations. If a non-cached route is needed, the node initiates a search procedure by flooding the network with route discovery packets. Upon reception of a discovery packet, the destination replies to the source with the routing information (e.g. path) collected by the discovery packet.

Route discovery can potentially cover a large part of the network, yielding a high flooding cost. In ad hoc networks, the flooding cost translates not only to delay and communication control overhead but also to consumption of node power resources. Minimization of transmissions is crucial for power conservation and extension of the overall network lifetime. The network uses expanding ring search and cooperative caching [2][3] to limit the flooding cost. In expanding ring search, each discovery packet is marked with a "Time To Live" (TTL) field, decreasing as the packet propagates through the network. If the destination is not found, a larger horizon route discovery restarts by issuing new discovery packets with increased TTL. The process continues until the destination is found. However,

a discovery can still be costly if the destination is located many hops away. Cooperative caching [2][3] advocates the use of any intermediate node that has cached the destination. Since routing information can be provided by non-destination nodes, the destinations each node decides to cache not only affects its own cost to discover a destination, but also the cost of other nodes in the network. The problem that arises is assigning the network caches subject to the memory limitations to minimize the overall network flooding cost.

We introduce a class of centralized anticipative route caching algorithms that yield near-optimal allocations subject to the node memory limitations. In addition to providing an upper bound on the performance of any distributed approach, the algorithms can be applied to scenarios where a central controller periodically collects topology and traffic information, computes the optimal cache contents, then transmits them back to the nodes. In military sensor network applications, this can be accomplished by Unmanned Air Vehicles (UAVs) flying over the sensor deployment area. After off-line computation, each sensor will always use the assigned cached routes while flooding for the non-cached destinations.

Route caching has been studied for Mobile Ad hoc Networks (MANETs)[5][6]. MANETs are characterized by a relatively small number of mobile nodes with high memory and processing capabilities. Even if a node has enough memory to cache information about all other nodes in the network, a problem arises when cached routes become invalid due to mobility. In a large static ad hoc network such as the one we consider, the issue is not stale cached routes but cache limitation with respect to network size. If a node discovers a new route and its cache is full, it must evict another route even if it is valid. Thus, the emphasis here is on cache placement policies rather than cache invalidation ones. In addition, while cooperative caching is optional in MANETs, in our context it is a key element for optimizing network performance.

The paper is structured as follows: In section II we formulate the problem and demonstrate that an optimal solution is infeasible for large network sizes. Section III introduces the Best-State/Best-Cost (BSBC) heuristic algorithms. The algorithm performance is evaluated in Section IV for various

system parameters. Finally, Section V concludes the paper.

II. PROBLEM FORMULATION

The wireless ad hoc network is represented as a graph $G(N, E)$. An edge $(i, j) \in E$ signifies that nodes i and j are within range. Each node i needs to address a set \tilde{D}_i of $M(\leq N)$ arbitrary destinations in the network. The popularity of destinations $j \in \tilde{D}_i$ is expressed by the route request probabilities $\mathbf{p}_i = [p_{ij}]$. A node can be both a source and a destination. The destination sets $\tilde{D} = [\tilde{D}_i]$ per source node, determine the source sets $\tilde{S} = [\tilde{S}_i]$ per destination node.

At any time, a node i can cache route information for only K out of the M destinations in \tilde{D}_i . Let ϕ be a network cache assignment. We define $D_i(\phi)$ to be the set of destinations cached by node i and $S_i(\phi)$ to be the set of nodes having cached routing information about node i as a destination. The complementary "non-cache" sets are $\bar{D}_i(\phi) = \tilde{D}_i - D_i(\phi)$ and $\bar{S}_i(\phi) = \tilde{S}_i - S_i(\phi)$.

When a need arises to route to a destination j that is not currently cached, i must perform an expanding ring search to discover routing information about this destination. Let $d(x, y)$ denote the minimum hop distance between nodes x and y . The routing information about destination j will be found at a minimum hop distance of:

$$d_{ij}(\phi) = \min\{d(i, j), \min_{k \in S_j(\phi)} \{d(i, k)\}\} \quad (1)$$

Given $d_{ij}(\phi)$, the flooding cost incurred by the expanding ring search is given by:

$$f_{ij}(\phi) = \sum_{d=1}^{d_{ij}(\phi)} n_i(d) \quad (2)$$

where $n_i(d)$ is the number of nodes covered when i floods at a maximum distance of d hops.

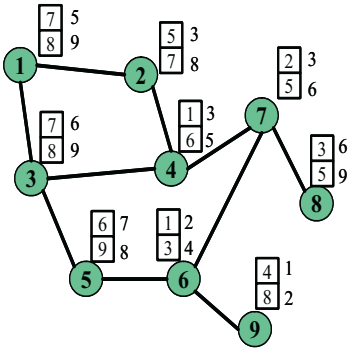


Fig. 1. A sample ad hoc network ($N = 9$) where each node addresses $M = 4$ arbitrary destinations but can cache only $K = 2$ of them. Routing probabilities are $p_{ij} = 1/M = 1/4$, $1 \leq i, j \leq 9$. A possible network cache state $\phi^{(t)}$ at time t is depicted. Node 1 has cached nodes $D_1(\phi^{(t)}) = \{7, 8\}$ out of $\tilde{D}_1 = \{5, 7, 8, 9\}$. The closest cache where 1 can find routing information about 9 is node 5 ($d_{19}(\phi^{(t)}) = 2$). According to expanding ring search, the flooding cost of 1 to acquire routing information for destination 9, is 7 hops ($f_{19}(\phi^{(t)}) = (2 + (2 + 3)) = 7$).

A performance indicator of the network caching ability is the flooding cost averaged over all nodes and route discoveries

for the non-cached destinations. We seek to minimize this cost over the entire duration of network operation. Long-term temporal behavior is captured by the route request probabilities p_{ij} . Given a cache assignment ϕ the long-term average flooding cost per route discovery is:

$$C_{avg}(\phi) = \frac{1}{N} \sum_{i=1}^N \sum_{j \in \bar{D}_i(\phi)} q_{ij}(\phi) \cdot f_{ij}(\phi) \quad (3)$$

where $q_{ij}(\phi) = p_{ij} / \sum_{k \in \bar{D}_i(\phi)} p_{ik}$ are the normalized route request probabilities of destinations j not cached by node i according to policy ϕ . The problem of interest can now be summarized as follows:

Problem statement: Given the spatio-temporal traffic demands (\tilde{D}_i, p_i) and cache limitation $K (< M)$ for each node i in the ad hoc network, find a cache assignment ϕ that minimizes the long term average flooding cost per route discovery.

The optimal solution can be found by performing an exhaustive search over all possible network cache assignments and selecting the allocation that yields minimum cost. Unfortunately, the number of assignments is $\left(\frac{M!}{K!(M-K)!}\right)^N$ which grows exponentially with the number of nodes in the network. This renders such a solution prohibitive for large network sizes— we must turn to heuristics.

III. BEST STATE / BEST COST (BSBC) HEURISTICS

The BSBC algorithms start from an arbitrary network cache state $\phi^{(1)}$ and converge to a "best state" after a number of iterations. During iteration t , node i computes a cost $c_i(j)$ for every destination $j \in \tilde{D}_i$ based on the current network cache state $\phi^{(t)}$. The node keeps the K maximum cost destinations in its cache and evicts the rest. The updated network cache state $\phi^{(t+1)}$ is determined by the new cache contents of node i . In the following iteration, the next node is considered and performs a caching decision based on $\phi^{(t+1)}$. The algorithms terminate when a "best state" has been reached. In such a state every node has cached maximum cost entries and the network cache state will not be further modified. The difference in the BSBC algorithms lies in the definition of the cost used to perform the caching decisions.

In the Local BSBC (L-BSBC) algorithm, node i decides by considering only its own weighted flooding cost for discovering each destination in \tilde{D}_i :

$$c_i(j) = p_{ij} \cdot f_{ij}(\phi^{(t)}), \quad \forall j \in \tilde{D}_i \quad (4)$$

In the Global BSBC (G-BSBC) algorithm, a node decides by taking into account the effect of its decision to the overall network cost. Given a network cache state ϕ , the total network cost for discovering destination j is the sum of the weighted costs of all source nodes in \tilde{S}_j that do not hold j in their cache:

$$G_j(\phi) = \sum_{k \in \bar{S}_j(\phi)} p_{kj} \cdot f_{kj}(\phi) \quad (5)$$

For each destination j , node i derives from $\phi^{(t)}$ two speculative network cache states ϕ_{j+} and ϕ_{j-} where j is present and absent from its cache respectively. The cost used for the caching decisions is:

$$c_i(j) = G_j(\phi_{j-}) - G_j(\phi_{j+}) \quad \forall j \in \tilde{D}_i \quad (6)$$

The motivation for this cost structure is that the K maximum speculative differences indicate the destinations with the greatest impact on the overall network cost for route discovery.

A. Complexity considerations

G-BSBC is expected to yield a lower cost than L-BSBC because it bases its decisions on the global network state. In L-BSBC each source i needs to compute only the cost to destination j . In G-BSBC node i needs to consider the sum of costs of all sources of j . Thus, the increase in computation complexity induced by G-BSBC per destination j given a network cache state $\phi^{(t)}$ is:

$$\rho^{(t)} = \frac{\sum_{j=1}^M \left(\overline{S_j^{(t)}} \right)}{\sum_{j=1}^M (1)} \quad (7)$$

This ratio can be very high for large M . A large M also implies a large set $\overline{S_j^{(t)}}$ of sources addressing destination j . The increase in computational complexity renders G-BSBC slower than L-BSBC and is the penalty for its better performance.

Being heuristics, both BSBC algorithms do not guarantee minimum cost at steady state. The following experiments demonstrate that they have a near-optimal behavior.

IV. PERFORMANCE EVALUATION

A. Experimental setting

The algorithms were tested on a 25×25 toroid mesh network ($N = 625$ nodes). Every node i addresses $M (\leq N)$ randomly located destinations, each destination j with equal probability ($p_{ij} = 1/M$). Node i can cache only $K (< M)$ destinations at any instant. To achieve independence on a specific locality of destination distribution, results for each parameter set (M, K) are averaged over 100 different spatial distributions.

We measure performance in terms of the long-term fraction of the network flooded per route discovery. Let ϕ^* be the cache state after a BSBC algorithm converges. As an average metric we use the long-term average number of nodes flooded per route discovery:

$$N_{avg}(\phi^*) = \frac{1}{N} \sum_{i=1}^N \sum_{j \in \tilde{D}_i(\phi^*)} q_{ij}(\phi^*) \cdot n_i(d_{ij}(\phi^*)) \quad (8)$$

This metric is similar to the cost $C_{avg}(\phi^*)$ of (eq. 3) where the total expanding ring search flooding cost $f_{ij}(\phi)$ has been replaced by its last term in eq. (2).

A worst-case metric is the maximum number of nodes flooded per route discovery at the BSBC steady state. This

equals the average over the maximum-cost destinations of each node:

$$N_{max}(\phi^*) = \frac{1}{N} \sum_{i=1}^N \max_{j \in \tilde{D}_i(\phi^*)} n_i(d_{ij}(\phi^*)) \quad (9)$$

The graphs illustrate the average $\frac{N_{avg}}{N} \times 100\%$ and worst case $\frac{N_{max}}{N} \times 100\%$ percentage of the network flooded per route discovery.

The performance of BSBC algorithms is compared to an ideal lower bound for the flooding cost. The bound assumes that, for every non-cached destination j , every source node i will discover routing information at the closest node in \tilde{S}_j . Due to the cache limitations this does not necessarily hold for every source and every destination. While not achievable, this bound serves as a useful reference point. The cost of the optimal solution lies between the cost of BSBC algorithms and the lower bound.

B. Effect of destination set size M

We first test the BSBC ability to cache an increasing part of the network using a fixed, small cache size. Figure 2 depicts the average performance when the cache size is fixed at $K = 10$ and M varies from 20 to 100 destinations.

For both BSBC algorithms, cost increases as M increases but the increase is minor after $M > 50$. Thus, a cache size of $K = 10$ can cache $M > 50$ destinations by flooding approximately 6% (L-BSBC) and 5% (G-BSBC) of the entire network on the average. The lower bound curve is not sensitive to the increase in M because no cache limitations are assumed in this case. For $M > 50$, both G-BSBC and L-BSBC have a constant difference from the ideal lower bound curve of about 3% and 4% of network coverage respectively.

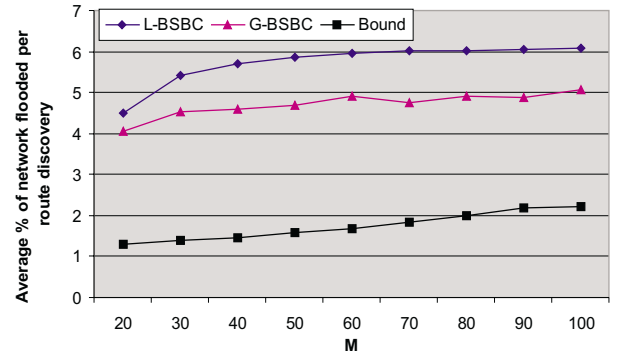


Fig. 2. Average Cost: $N = 625$, $K = 10$, M varies.

G-BSBC performs better than L-BSBC by yielding approximately 1% less network coverage for $M > 30$. Better performance is expected since G-BSBC performs caching decisions by taking into account the entire network state. However, the relative performance of the algorithms is reversed when we consider the worst-case metric depicted in Figure 3. This counter-intuitive phenomenon is due to the difference in the algorithm cost structures. According to L-BSBC, a node places

in its cache the K maximum cost destinations with respect to itself. In G-BSBC the maximum cost destinations placed in a cache are the ones that minimize overall network flooding cost. Hence, it is possible for a node using G-BSBC to decide not to place the maximum cost destination with respect to itself in its own cache. The worst-case metric reflects the maximum cost of all destinations per individual source node and therefore favors L-BSBC.

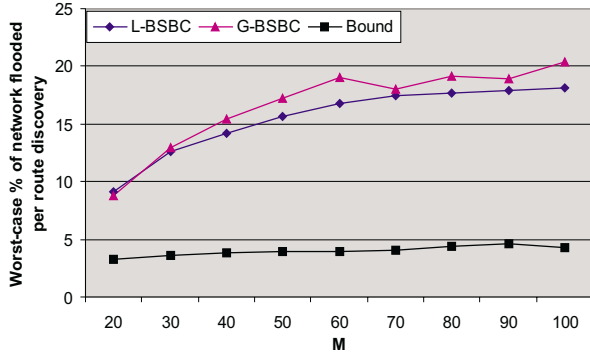


Fig. 3. Worst-case cost: $N=625$, $K=10$, M varies.

C. Effect of cache size K

In this experiment, we varied the Cache size K , while keeping M fixed to 100 destinations. As the cache size increases, more destinations can be cached and the cost is expected to decrease. Average performance is illustrated in Figure 4. G-BSBC performs better than L-BSBC, as expected. The maximum difference in cost is 1% and is for $K = 10$. The difference decreases as K increases. For $K > 40$ the algorithms perform similarly. Both algorithms converge to the lower bound for $K > 50$. Thus, caching half the destinations is adequate for providing a close to optimal network flooding cost on the average. In the worst-case (Fig. 5), G-BSBC and L-BSBC perform similarly for $K > 10$. For larger cache sizes there are less non-cached entries and the worst-case metric does not favor L-BSBC any more. Both Figures 4 and 5, are

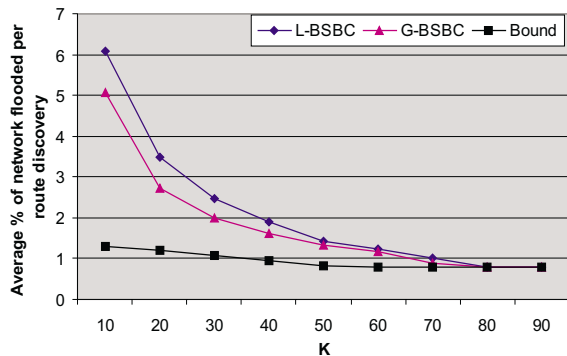


Fig. 4. Average Cost: $N = 625$, $M = 100$, K varies.

characterized by a "transient" and a "steady-state" region. In

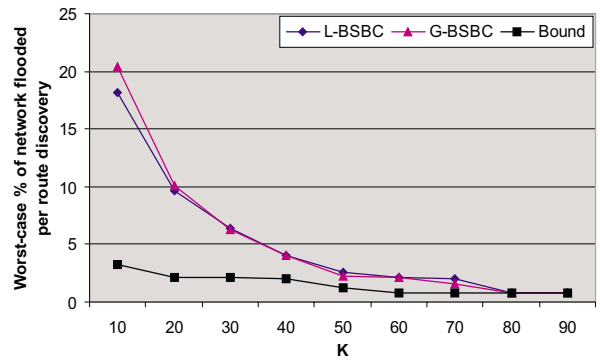


Fig. 5. Worst-Case Cost: $N = 625$, $M = 100$, K varies.

the transient region a small increase in cache capacity yields a dramatic decrease in flooding cost. The transient region ($10 \leq K \leq 50$) reveals the rate at which flooding cost decreases as cache capacity increases. A faster decrease in cost for the same increase in K indicates the algorithm that better exploits the extra cache positions. Note that in the lower bound curve no transient region exists since there are no cache limitations in this case.

The steady-state region is characterized by a threshold above where no major improvement in flooding cost exists. When comparing two caching algorithms, the smaller this threshold, the better the algorithm. The graphs indicate this threshold to be $K = 50$ for both L-BSBC and G-BSBC. For $K \geq 70$ both algorithms converge to the lower bound curve. This cache size is sufficient to cache the entire network in terms of both the average and worst-case metrics.

D. Special case: Comparison to the optimal solution

In this experiment, we demonstrate the near-optimal performance of BSBC for a special case where the optimal solution can be computed. Consider a toroid mesh network where each node addresses all other nodes ($M = N - 1$) with equal probability ($p_{ij} = 1/(N - 1), \forall i, j \in N$). Due to the uniform traffic demands and the symmetric wrap-around structure of the toroid mesh, every node can be treated uniformly. Given any destination node and a cache capacity K , placing the "aware" nodes caching this destination in a diamond structure around the destination is the strategy that yields the minimum flooding cost (established in [8]). In addition, by the symmetry of the optimal solution the cache size K of each node equals the number of "aware" nodes for a specific destination. These arguments determine a way of assigning destinations to the cache of every node in an optimal manner.

In the 625-node toroid mesh of Figure 6, every node addresses all other nodes in the mesh network ($M = N - 1 = 624$) with probability $1/624$ and can cache only $K = 40$ of them. Without loss of generality, the destination considered is the center node. According to the optimal strategy, the nodes aware of this destination must be placed in a diamond structure surrounding it. The rest of the nodes use the caches of the aware nodes to find the routing path for this destination.

Also the number of aware nodes is equal to the cache size of every node in the network (40). In this experiment we set $M = N - 1 = 624$, $K = 40$ and $p_{ij} = 1/624$ and ran the L-BSBC algorithm on the 625-node toroid mesh structure of Figure 6. After convergence, we placed on the mesh the nodes that have cached information about the center node.

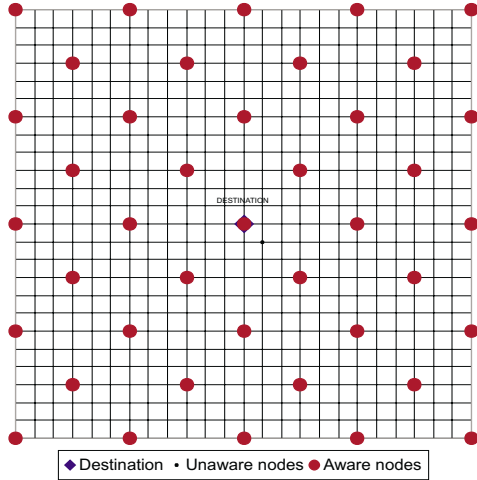


Fig. 6. Optimal: $M = N = 625$, $p_{ij} = 1/625$, $K = 40$: The optimal cache assignment for the destination node (center) is to cache it at nodes arranged in a diamond structure. Expected coverage is 1.93%.

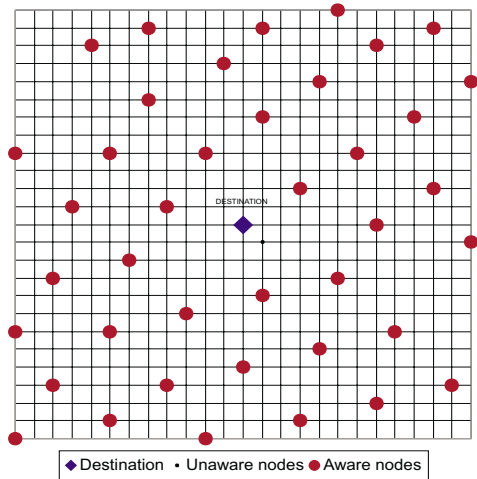


Fig. 7. L-BSBC: $M = N = 625$, $p_{ij} = 1/625$, $K = 40$: Nodes that keep routing information about the destination node (center) are placed in a diamond-like pattern. Expected coverage is 1.99%.

According to Fig. 7, L-BSBC places the aware nodes in a diamond-like pattern. The optimal diamonds strategy yields an average (and worst-case) cost of 1.93% while L-BSBC yields 1.99%. Thus, for the special case of a toroid mesh network and uniform traffic, the BSBC algorithm yields a near-optimal solution.

V. CONCLUSIONS

We introduced the class of BSBC algorithms to address the problem of optimal route caching in large, memory-limited

ad hoc networks. These algorithms take into account spatial and temporal traffic demands; they result in cache assignments targeting minimization of the expected flooding cost per route discovery. Both BSBC algorithms demonstrated near-optimal performance. G-BSBC performs slightly better than L-BSBC at the expense of increased computational complexity.

The BSBC algorithms can be directly applied to the design of unattended static ad hoc networks where information on cache limitations and traffic estimates are provided in advance to the designer. Sensor networks and the Ricochet network [4] fall in this category. Minimizing flooding cost for route discovery is central to the maximization of the lifetime and the performance of these networks.

The BSBC algorithms can also provide a direct measure for performance evaluation of real-time distributed caching algorithms. Indirect measures such as end-to-end packet delivery ratio and end-to-end delays have been considered in [5][6][7]. A direct measure of the network caching efficiency is the expected network flooding cost per route discovery $C_{avg}(\phi)$ (eq. 3). The performance of a distributed real-time caching algorithm can be evaluated as follows: at certain sampling time instants t , we freeze the mobile ad hoc network and compute $C_{avg}(\phi^{(t)})$ based on the current network cache state $\phi^{(t)}$. We then run a BSBC algorithm on the current topology and \tilde{D} , the spatial demands¹. The BSBC cache assignment will yield a lower cost than the current network cache assignment induced by the distributed algorithm.

Our results suggest that future real-time distributed caching algorithms for mobile networks take into account partial neighborhood caching state and traffic estimates to minimize flooding cost. Toward this effort, the BSBC algorithms are the first step taken to address the problem of optimal route caching under memory limitations and provide a useful reference for future distributed approaches.

REFERENCES

- [1] Park, M. S. Corson, *A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks*. Proceedings INFOCOM '97, Kobe Japan, 1997.
- [2] D. Johnson and D. Maltz., *Dynamic Source Routing in mobile ad-hoc networks*. In Mobile Computing, edited by T. Imielinski and H. Korth, chapter 5, pages 153-181, Kluwer Academic Publishers, 1996.
- [3] C. Perkins, E. Royer, *Ad Hoc On Demand Distance Vector Routing*. Proceedings IEEE Workshop on Mobile Computing Systems and Applications, 1999.
- [4] Ricochet Networks, Inc., *The Ricochet project*. <http://www.ricochet.net/>.
- [5] Y. Hu, D. Johnson, *Caching strategies in On-Demand Routing Protocols for wireless ad hoc networks*. Proceedings ACM Mobicom '00, Boston 2000.
- [6] M. Marina and S. Das, *Performance of Route Caching Strategies in Dynamic Source Routing*. Proceedings International Workshop on Wireless Networks and Mobile Computing (WNMC) '01, April 2001.
- [7] J. Broch, D. A. Maltz, D. B. Johnson, Yih-Chun Hu, and J. Jetcheva, *A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols*. Proceedings ACM Mobicom 1998.
- [8] T. Salonidis, *Optimal route caching in large wireless ad hoc networks*. Technical Report MS99-13, Institute of Systems Research (ISR), University of Maryland.

¹The demands \tilde{D} may be known at the beginning of the network operation; alternatively, we may assume $M=N$, i.e. every node addresses all other nodes in the network.