Hash-Based Dynamic Source Routing

Claude Castelluccia and Pars Mutaf

INRIA Rhône-Alpes
ZIRST - 655, Avenue de l'Europe
38334 Saint Ismier Cedex - France
{claude.castelluccia, pars.mutaf}@inria.fr
http://www.inrialpes.fr/planete.html

Abstract. This paper [1] presents and evaluates *Hash-Based DSR*, a *DSR* extension for large networks. This protocol reduces the per-packet control overhead of DSR by compressing the source-route with a Bloom filter. Simulations on large networks show that $HB - DSR$ increases the network capacity by a factor of up to 15. $HB - DSR$ is an attractive extension to DSR for large ad-hoc networks.
Another important property of $HB - DSR$ is that, as opposed to DSR, its performance is similar for IPv4 and IPv6. While IPv6 large addresses are prohibitive in DSR, we show by simulations that $HB - DSR$ performs as well for both IP versions. This is important contribution considering the growing interest of the wireless network community for IPv6.

1 Introduction

In an *ad hoc wireless network*, mobile nodes communicate with each other using multi-hop wireless links. Each node is also a router and is therefore part of the routing infrastructure. The bandwidth and the nodes' capacity (processing and power) of such networks are usually very limited. Therefore one of the biggest scientific challenges of this area is to design routing protocols that minimize the control overhead, i.e. the bandwidth overhead used to establish and maintain the routes, as much as possible.

Several protocols have been proposed recently. One of them is the *Dynamic Source Routing* (DSR) protocol [2]. The main characteristic of DSR is that it uses source-routing to route packets from the source to the destination. *DSR* is probably one of the most efficient protocols. However it has two important limitations:

1. *Scalability limitation*: *DSR* uses source-routing and, as a result, does not scale to large networks.
2. *IPv6 unfriendly*: IPv6's large address space limits considerably the performance of DSR. This is an important drawback considering the growing interest of the wireless community for IPv6.

[1] A full version of this paper is available as Technical Report INRIA-4784 [1].

Our goal is to propose an extension to DSR that overcomes these two problems, while still preserving its current features. We propose *Hash-Based DSR*, a protocol that compresses the list of addresses in the source-route using a Bloom filter [3]. Instead of inserting a source-routing option in each packet, as in DSR, the source inserts the corresponding Bloom filter.

The rest of the paper is organized as follows: Section 2 presents the proposed *Hash-Based DSR* protocol. In Section 3, some simulation results are presented and analyzed. Section 4 presents some related work. Finally, the last section concludes the paper.

2 Hash-Based DSR

One of the biggest limitations of DSR for large network is that it uses source-routing to route packets. A DSR packet must carry the IP addresses of all the nodes that are on the path from the source to the destination. The generated bandwidth overhead is significant when the source and the destination are far from each other. Indeed as we will show later in our simulations (see Figure 5), this control overhead can use up to 95% of the network bandwith when the path length is about 100 nodes. This overhead is not acceptable and limits the use of DSR to small networks.

We propose, in this paper, *Hash-Based DSR* (HB-DSR) a DSR extension for large ad-hoc networks. The main idea of our scheme is to compress the list of addresses in the source-route using a Bloom filter. Instead of inserting a source-routing option in each packets as in DSR, the source inserts the corresponding Bloom filter. The resulting protocol is very similar to DSR and inherits from most of its features. Furthermore since the filter used in each packet is much smaller that the list of addresses, $HB-DSR$ is much more scalable than DSR for large networks.

2.1 Bloom Filters

A Bloom filter is a m-bit vector v that codes the membership of a set $A = \{a_1, a_2, .., a_n\}$ of n elements [3]. The idea is to allocate a vector v of m bits, initially all set to 0, and then to choose k independent hash functions, h_1, h_2,...,h_k, each with a range $\{1, ..., m\}$. For each element a of A, the k bits at positions $h_1(a)$, $h_2(a)$, ...,$h_k(a)$ in v are set to 1.

To verify whether a element b is the set A, it is enough to verify if all the k bits at positions $h_1(b)$, $h_2(b)$, ...,$h_k(b)$ in v are set to 1. If any of them is set to 0, b does clearly not belong to A. If all the bits are set to 1, then b is probably a member of A, although there is some probability that b is not a member of A. This is called a *false positive*. The probability a *false positive* can be calculated in a straightforward fashion. After all the elements of A are coded in the Bloom filter, the probability that a specific bit is still set to 0 is: $(1 - 1/m)^{kn}$ The probability of a false positive is then:

$$fp = (1 - (1 - 1/m)^{kn})^k \tag{1}$$

It can be shown that for a given m and n, the optimal value of hash functions to used, k, is $k = ln2 \times m/n$. There is a clear trade-off between the size of the filter m and the probability of a false positive. For a given n, fp can be decreased by increasing m. However increasing m reduces the compression rate of the Bloom filter. The optimal value of m is application-specific. For some applications, the false-positive rate must be very small and as a result m must be large. Other applications can tolerate higher false positive rates and therefore can use smaller values of m.

2.2 Protocol Overview

In $HB - DSR$, the DSR source-routing option is *compressed* using a Bloom filter. The set A is therefore composed of the nodes' IP addresses of the path from the source to the destination. n is equal to the path length. The proposed protocol is the following:

1. A source S that wants to send packets to a destination D, invokes the *DSR route discovery* protocol. It then receives a *Route Reply* that contains the list of the nodes' addresses along the path from S to D.
2. S then computes from these addresses the corresponding Bloom filter as follows:
```
1: for ($j = 1; j < plen; j++$) do
2:   for ($i = 0; i < k; i++$) do
3:     $BF[hash(i|Addr_j).mod(m)] = 1;$
4:   end for
5: end for
```
where the Bloom filter, BF, is a bit-string of size m, k is the number of hashes used and $Addr_j$ are the ($plen - 1$) addresses on the path from S to D [2].
3. When S sends packets to D, it inserts a (newly defined) hop-by-hop option that carries the bloom filter and the value of the parameter k (the number of hashes to use).
4. Upon receiving a packet, a node verifies whether the destination address is one of its addresses. If this is the case, the packet has reached its destination. If destination address is not one of its addresses and the TLL is zero, the packet is dropped, otherwise the node verifies if any of its neighbors' addresses (except the one it received the packet from) are contained in the Bloom filter carried in the packet using the following algorithm:
```
1: for ($j = 1; j <= R; j++$) do
2:   $IsMember[j] = 1;$
3:   for ($i = 0; i < k; i++$) do
4:     if ($BF[hash(i|addr_j).mod(m)] == 0$) then
5:       $IsMember[j] = 0;$
6:     end if
7:   end for
```

[2] The following sections describe how the parameters m and k are chosen.

```
8: end for
```

where R is the number of neighbors N_j defined by their address, $addr_j$.
If, when the algorithm terminates $IsMember[j]$ is set to 1 then the neighbor N_j belongs to the filter and is therefore a node on the path from S to D, otherwise N_j does not belong to the filter. If a neighbor belongs to the Bloom Filter, the packet's TTL is decremented and the packet is forwarded to that neighbor. If not, the packet is silently dropped (the packet was probably mis-routed from a false positive). If there are several neighbors that are contained in the packet's filter (this is the result of false positives), the packet is duplicated and forwarded to each of these neighbors.

As a result of this protocol, the packets are forwarded hop-by-hop until the destination.

The filter's size is obviously a crucial parameter of our protocol. In fact if the filter's size is too small, false positives will be frequent and the packets will be mis-routed. Although the packets will reach their destination, the bandwidth overhead will be quite large. On the other hand, if the filter's size is very large, false positives will be rare (and therefore no many packets will be mis-routed) but the size of the packets will be larger (since they carry the filter) and the benefit of the compression will be reduced.

2.3 Filter Size Computation

This section presents two different approaches to compute the size of the Bloom filter. In the first one (open-loop algorithm), a mobile host computes the "optimal" size using some cost function. This computation is performed locally without any feedback from the network. In the second approach (closed-loop algorithm), a host computes the filter size using some inputs he gets from the network. Both approaches can be used conjointly.

Notation. In this section, we define and use the following notation:
- *plen:* the path length from source to destination.
- *R:* the average number of neighbors per intermediate node.
- *fp:* the false positive probability.
- *IP_addr_size*: the size of an IP address (32 bits for IPv4 and 128 bits for IPv6).
- *IP_head_size*: the size of an IP header.
- *IP_opt_size*: the size of the IP option header that carries a Bloom filter.
- *pkt_size:* the total size of a packet including its IP header, options and its data.
- *m*: the size of the Bloom filter.

Open-loop Algorithm. In DSR, the routing overhead resulting from the source routing is defined by:

$$dsr\_cost = plen \times [(plen - 1) \times IP\_addr\_size + IP\_opt\_size] \qquad (2)$$

In HB-DSR, the routing cost results from (1) the Bloom filter that each packet carries and (2) the false positives. If a packet creates a false positiveat a node,

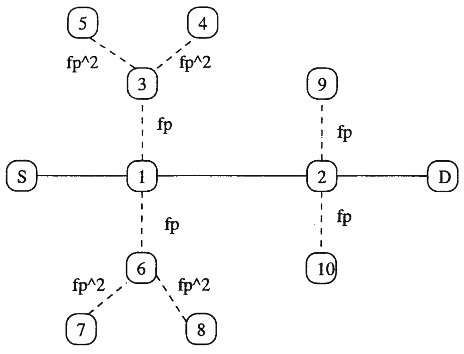


Fig. 1. HB-DSR routing

the packet is sent to one or several wrong paths. Let us consider the scenario displayed in Figure 1. In this figure, S is sending packets to D. The path length, *plen* is equal to 3.

At the first node, *N1*, the probability that a false positive happens towards N3 (resp. N6) is defined by *fp*. The resulting cost is $fp \times pkt\_size$. The probability that the forwarded packet is forwarded to N4 (resp. N8) or to N5 (resp. N7) is fp^2. The induced cost is then $fp^2 \times pkt\_size$. The probability that the forwarded packet is forwarded by N4 or N5 is zero because the TTL has then reached zero. The total cost at node N1 is therefore:

$$Cost_{N1} = 2 \times pkt\_size \times [fp + (R-1) \times fp^2] \tag{3}$$

Similarly at the second node, N2, the probability that a false positive happens towards N9 (resp. N10) is *fp*. The resulting cost is $fp \times pkt\_size$. The probability that this packet is forwarded by N9 (resp. N10) is zero because the TTL is then 0. The total cost at node *N2* is then:

$$Cost_{N2} = 2 \times pkt\_size \times fp \tag{4}$$

More generally, the cost at node Nx (with $0 < x < plen$) is defined by:

$$Cost_{Nx} = (R-2) \times pkt\_size \times fp \times \sum_{i=0}^{plen-x-1} ((R-1)^i \times fp^i) \tag{5}$$

As a result, the total cost resulting from the false positives by a packet sent from a source to a destination is defined as follows:

$$fp\_cost = \textstyle\sum_{j=1}^{plen-1} Cost_{Nj}$$

$$= (R-2) \times pkt\_size \times fp \times [\textstyle\sum_{j=0}^{plen-2}((plen-j-1) \times (R-1)^j \times fp^j)] \tag{6}$$

where *fp* is computed as follows:

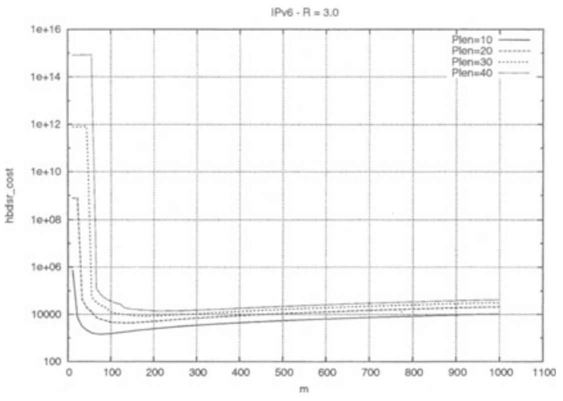


Fig. 2. HB-DSR cost vs m

$$fp = (1/2)^{ln(2) \times m/plen} \tag{7}$$

The total HB-DSR overhead is the cost resulting from the false positive and the cost of carrying the filter in each packet, i.e:

$$hbdsr\_cost = plen \times (IP\_opt\_size + m) + fp\_cost \tag{8}$$

Figure 2 plots the *hbdsr_cost* function according to m for several path lengths ($plen$). This figure shows that, for a given path length, there is only one value of m that minimizes the cost function.

In HB-DSR, when a source wants to send packet to a destination it performs a DSR route discovery. It then receives a Route Reply that contains the list of addresses along the path. It can deduce from this information the path length ($plen$) but still need to estimate the parameter R. We propose to extend the route request and reply messages with a *CN (Cumulative Neighbors) field.* The field is set to 0 in the route request by the source. Each intermediate node that forwards it increments this CN field by the number of its neighbors. When the destination receives the route request, it copies the CN value in the route reply and sends it to the source. The source can then compute an estimate of the parameter R by dividing the value CN by the path length ($plen$).

Once the source has an estimate of $plen$ and R, it computes for several m the value of the cost function *hbdsr_cost* and uses the value of m that minimizes it.

Closed-loop Algorithm. In the previous section, the source computes the filter size using a cost function. However in some scenarios, the source can still suffer from the cost of false positives. An closed-loop algorithm is often more efficient and practical. We therefore propose to use some feedbacks from the network to tune the filter size.

We define, *FP_DUP*, a new error message. When a node, N_i, detects one or several false positives (i.e. there are more than one neighbor, let's say r_i neighbors, in the filter), it returns a *FP_DUP* message to the source. This message contains the number of neighbors contained in the filter, i.e. r_i, at node N_i.

In order to avoid the explosion of *FP_DUP* messages and to avoid routing loop, we propose that nodes drop packets that have experienced two or more false positives. When a node detects a false positive upon reception of a packet, it sends a *FP_DUP* to the source and set a bit (the D bit) in the packet to 1. If this bit is already set, the packet is dropped and, instead of a *FP_DUP* message, a *FP_DROP* message is sent to the source.

The source executes the following algorithm:

- *Step1*: If the source receives one or several *FP_DROP* messages, it increases the filter size by one unit (i.e as shown in Section 3 by 64 bits for IPv6 or 32 bits for IPv4). The algorithm is then re-executed.
- *Step2*: If the source receives one or several *FP_DUP* (but no *FP_DROP* message) that specify N false positives, the source computes *FP_cost*, the cost resulting from the N false positives:

  $$FP\_cost = N \times pkt\_size \tag{9}$$

  It also computes, *Delta_cost*, the cost resulting from increasing the filter size by one unit, BF_unit.

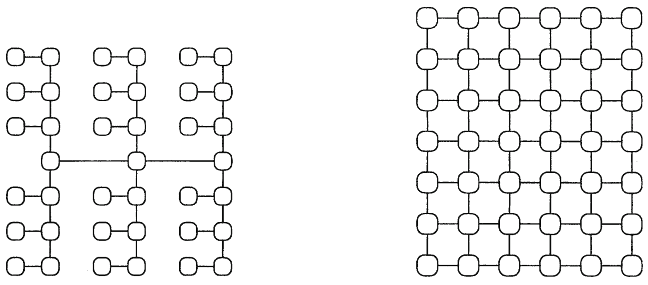
  $$Delta\_cost = plen \times BF\_unit \tag{10}$$

  If $Delta\_cost < FP\_cost$ then the filter's size is incremented by one unit and the algorithm is re-executed. Otherwise the filter size is kept to m and the algorithm terminates.
- *Step3*: If the source does not receive any *FP_DUP* nor any *FP_DROP* messages for a given value of m and if this m was not obtained from *Step2*, the source decrements m by one unit and the algorithm is re-executed. Otherwise the algorithm terminates.

3 Simulation

3.1 Simulation Model

The main goal of our simulations is to evaluate the gain of our approach over DSR. We mainly focus our simulations on the per-packet bandwidth gain achieved by compressing the source-routes with Bloom filters. We do not consider the control messages (such as RREP, RREQ or RRER) overhead since they are identical in DSR and $HB - DSR$. Furthermore, for simplicity, we do not consider mobility in our simulations. The main contribution of our scheme is to reduce the per-packet control overhead. All the rest, including mobility management, is similar to DSR. We expect the mobility management performance of DSR and HB-DSR to be very similar.



(a) $MANET_1$ (b) $MANET_2$

Fig. 3. Network topology

We considered two different network topologies (see Figure 3). Both of them contain 3600 nodes. The topology of the first one ($MANET_1$) is a tree and as a result there is only one possible path from a given source to a given destination. The second one ($MANET_2$) is highly connected (a node has on average 4 neighbors while in $MANET_1$ a node has on average 2.8 neighbors) and there are several paths for a given source to a given destination.

For each of the network, we randomly select the source and the destination. We then simulate the routing of packets from the source to the destination using DSR and the 3 following variants of $HB - DSR$:

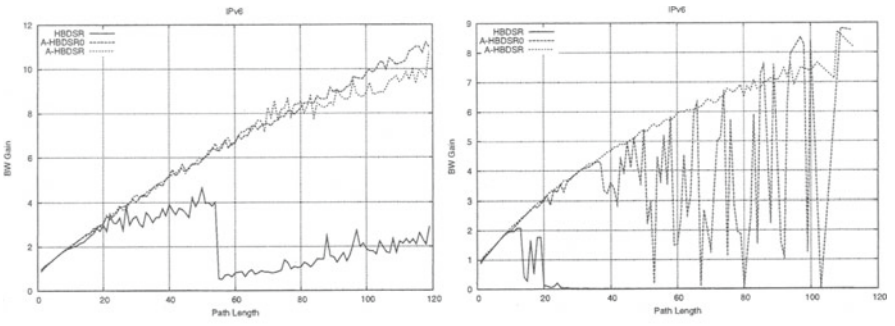
1. $HB - DSR$: the filter size is constant (88 bits for IPv6).
2. $AHB - DSR_0$ (A stands for *Adaptive*): the filter size is computed using the cost function described in Section 2.3.
3. $AHB - DSR$: the filter size is computed using the cost function described in Section 2.3 and adjusted using feedbacks from the network (FP_DUP and FP_DROP messages), as described in Section 2.3.

We run these simulations 2500 times for IPv6 and IPv4.

3.2 Performance Results

Figure 4 displays the *bandwidth gain* of $HB - DSR$, $AHB - DSR_0$ and $AHB - DSR$ for IPv6 according to the path length between the source and the destination. We compute the *bandwidth gain* of a scheme S by dividing the bandwith used by DSR to transmit a packet containing 64 bytes of data by the bandwidth used by the scheme S to transmit the same data to the destination. Each hop-wise transmission is counted as one transmission. We assume here that the route discovery phase has been performed and that the source knows the route to the destination.

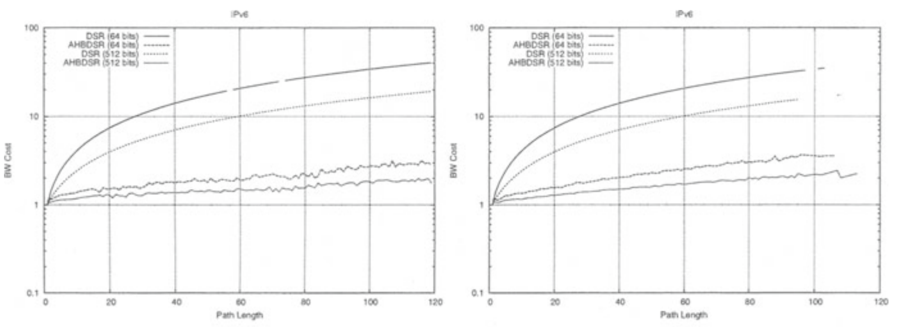
The results show that:



(a) $MANET_1$ (b) $MANET_2$

Fig. 4. Bandwidth gain (IPv6; data-size = 64 bytes)

- $HB-DSR$: When m is fixed and equal to 88, the gain decreases considerably with the path length. These results were expected because when the path length is large, the false positive rate is close to 1 and most of the transmitted packets are broadcast. For $MANET_1$, the gain is always greater than one because the network is not highly connected and the broadcast cost is not too high (and a least lower than source-routing). In contrast, for $MANET_2$, the gain converges to 0. $MANET_2$ is highly connected and the cost of broadcasting a packet is very large. It is more efficient to use source-routing, i.e. regular DSR, in this scenario.
- $AHB - DSR_0$: The performance for $MANET_1$ and $MANET_2$ are very different. With $MANET_1$ the gain is always greater than one and increases with the path length. When the path length is 100 nodes, the achieved gain is 10. This means that the network can accommodate 10 times more connections than if DSR was used. With $MANET_2$, $AHB-DSR_0$ does not perform well and its performance fluctuates a lot. These results are explained by the different costs of a false positive in each networks. In $MANET_1$, the cost of a false positive is not too high because the network is not very well connected and the mis-routed packets die out quickly. In $MANET_2$, the false positive cost is quite high. In fact, since each node has a higher number of neighbors, the probability of a false positive is larger and as a result, false positives are more frequent. Furthermore, if a packet is mis-routed twice in a row, it can reach the correct path again and enter a routing loop.
- $AHB-DSR$: $AHB-DSR$ corrects the problems of the $AHB-DSR_0$. By using feedbacks, the source can adjust the value of m accurately and minimizes the number of mis-routed packets. The gain increases with the path length. The gain obtained with $MANET_1$ is larger than the gain achieved with $MANET_2$. Indeed, since the connectivity is smaller, fewer bits are required per bloom filters in $MANET_1$.



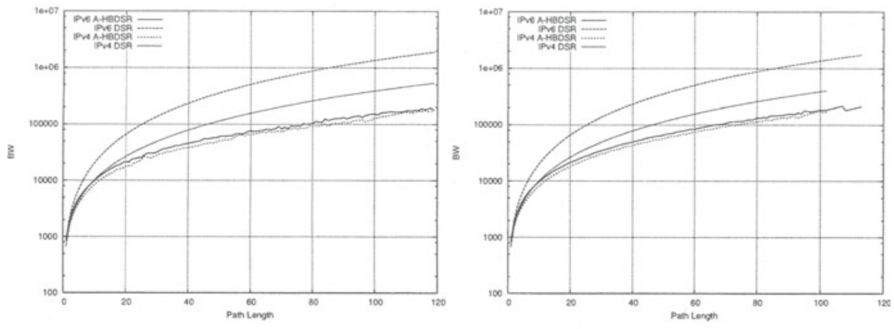
(a) $MANET_1$ (b) $MANET_2$

Fig. 5. DSR and AH-BDSR bandwidth cost (IPv6)

Figure 5 presents the *bandwidth cost* of $AHB - DSR$ and DSR. The bandwidth cost of a given proposal is computed by dividing the number of bytes necessary to transmit a given piece of data from the source to the destination using the proposal by the number of bytes necessary to transmit the same data with regular IP. This cost actually measures the control cost resulting from carrying in each packet a source-route or a Bloom Filter. We consider two data-sizes: 8 and 64 bytes. The results show that DSR cost increases drastically with the path length. In fact since a packet carries the addresses of all the nodes from the source to the destination, the control cost increases with the path length. When the path length is 100, the cost of DSR for a data size of 64 bytes is 20. This means that the cost of sending 64 bytes from the source to the destination is 20 times larger than the cost needed to transport the same data in a regular IP packet (i.e. without the source-routing option). The cost to transport these data with $AHB - DSR$ goes down to 3. This is the result of compressing the source-address with a Bloom filter. As we will see later in this section, this cost is much lower when the data size is larger.

Figures 6 compare the performance obtained with IPv4 and IPv6. They display the bandwidth (in bytes) used by DSR and $AHB - DSR$ to transmit 64 bytes of data from the source to the destination with IPv4 and IPv6. Each hop-wise transmission is counted as one transmission. These results show:

1. DSR is more expensive in IPv6 than in IPv4. In fact since addresses are much larger in IPv6 than in IPv4, the source-routing overhead is much larger. As a result, IPv6-DSR is more expensive than IPv4-DSR in term of bandwidth.
2. IPv4 and IPv6 $AHB - DSR$ costs are very similar. The filter's size used by $AHB - DSR$ is independent from the IP version. The resulting cost is the same despite large IPv6 address size. This result is very encouraging. While IPv6 is an handicap for DSR, it becomes very attractive for $AHB - DSR$.



(a) $MANET_1$ (b) $MANET_2$

Fig. 6. IPv4 vs IPv6 bandwidth (data-size= 64 bytes)

4 Related Work

Bloom filters have recently received attention in the networking area. A complete survey of these proposals can be found in [4]. [5] introduces a longest prefix matching algorithm that deploys "counting Bloom filters", in order to efficiently narrow the scope of the search and hence increase the IP lookup speed. [6] proposes using Bloom filters to reduce the storage cost of interface lists held by routers for group communications. Instead of maintaining a list of interfaces for each group, for each interface a Bloom filter is maintained. If a group is active with respect to an interface, then the corresponding Bloom filter is inserted the group. False positives, in this case, lead to some packets being forwarded incorrectly, which will be eventually discarded by other nodes.

In another related work we have proposed using Bloom filters for reducing the broadcast cost of paging in IP-based cellular systems [7]. IP paging is an optimization that allows the mobile IP hosts to preserve energy by entering dormant mode within the boundaries of a wide paging area. Using "hash-based paging" the paging sub-system inserts the IP addresses of the called dormant hosts (in the same paging area) into a Bloom filter, and page them concurrently by broadcasting a single paging message. Similarly to $HB-DSR$, hash-based IP paging is more attractive for IPv6, since a paging message that contains a number of 128-bit IPv6 addresses would consume too much bandwidth.

The proposed optimization has some relevance with header compression, but in a more general sense. The term "header compression" mostly implies techniques such as [8][9][10]. These techniques are based on the observation that most TCP/IP header fields never or seldom change during a session, and can easily be compressed. The proposed optimization is more a compression technique that is specific to *a list of IP addresses.* Compression can be made in a stateless fashion. Decompression is replaced by membership query, which is also stateless.

This allows $HB - DSR$ to compress source-routes while avoiding routing state (or, routing context) at intermediate nodes.

5 Conclusions

We have presented and evaluated *Hash-Based DSR*, a DSR extension for large networks. This protocol reduces the per-packet control overhead of DSR by compressing the source-route with a Bloom filter. The simulation results on large networks (diameter of 100 nodes) show that $HB - DSR$ increases the network capacity by a factor of up to 15.

Another important benefit of $HB - DSR$ over DSR is that its performance is similar for IPv4 and IPv6. In fact while IPv6 large addresses are prohibitive in DSR, we showed by simulations that $HB - DSR$ performs as well for both IP versions. We expect this result to generate a lot of interest in the IPv6 community.

$HB - DSR$ can be extended to support *one-to-many* communication. In fact, if a source knows the source-routes to each of its destinations, it can build the Bloom filter that contains all the nodes of the delivery tree. By inserting such filter in a packet, the packet will be routed from the source to its destinations using the optimal delivery tree. We believe that this approach is promising for small groups.

References

1. Castelluccia, C.: Hash-Based Dynamic Source Routing. Technical Report 4784, INRIA (2003)
2. Johnson, D., Maltz, D., Broch, J.: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks. In Perkins, C., ed.: Ad Hoc Networking. Addison-Wesley (2001) 139–172
3. Bloom, B.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13** (1970) 422–426
4. Broder, A., Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey. In: Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing. (2002)
5. Dharmapurikar et al, S.: Longest Prefix Matching Using Bloom Filters. In: Proceedings of SIGCOMM'2003 Conference. (2003)
6. Gronvall, B.: Scalable multicast forwarding. SIGCOMM Poster (2001)
7. Mutaf, P., Castelluccia, C.: Hash-based Paging and Location Update Using Bloom Filters. to appear in ACM Mobile Networks and Applications (MONET) **10** (2005)
8. Jacobson, V.: Compressing TCP/IP Headers for Low-Speed Serial Links. RFC 1144, IETF (1990)
9. Degermark et al, M.: IP Header Compression. RFC 2507, IETF (1999)
10. Bormann et al, C.: RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed. RFC 3095, IETF (2001)