ZigZag: A Content-based Publish/Subscribe Architecture for Human Networks

Yaxiong Zhao and Jie Wu
Department of Computer and Information Sciences
Temple University
Philadelphia, PA 19122, USA
{yaxiong.zhao, jiewu}@temple.edu

***Abstract*—The wide use of mobile devices gives rise to many novel network applications, which require an efficient communication substrate. These networks, which are comprised solely of human-carried mobile devices, are called *human networks* (HUNETs). In this paper, we present *ZigZag*, a *content-based publish/subscribe* (CBPS) architecture for HUNETs. We deem that CBPS is a natural fit for HUNETs for its inherent anonymity, scalability, and flexibility. However, existing CBPS systems are designed for the Internet, consuming resources beyond the capacity of mobile devices. Additionally, they do not consider user mobility. To address these restrictions, ZigZag employs a transformation-based subscription representation and processing technique, and a Bloom-filter-based message routing and forwarding scheme. The advantages of ZigZag are its low subscription storage consumption and matching complexity. Analytic and simulation results prove that our proposed methods are flexible and more efficient than existing techniques.**

***Index Terms*—Bloom filter, human networks, content-based publish/subscribe, Z-order curve**

I. Introduction

The wide use of portable mobile devices and the fast development of wireless communication technologies have enabled individuals the ubiquitous accessibility to network resources. Previously, such devices were connected through wireless infrastructures, which only provide limited services. Recent research on *delay tolerant networks* (DTNs) [12] provides a new paradigm for the communication of intermittently-connected wireless devices without the aid of an infrastructure. In this paper, we take a step further and incorporate application requirements into the network design. We propose *human networks* (HUNETs), a new application platform working on top of the networks, formed solely by human-carried wireless devices. An real-world example is Bluejacking [3], where people use bluetooth devices to exchange messages when they are in vicinity. Fig. 1 illustrates an example HUNET of 5 users.

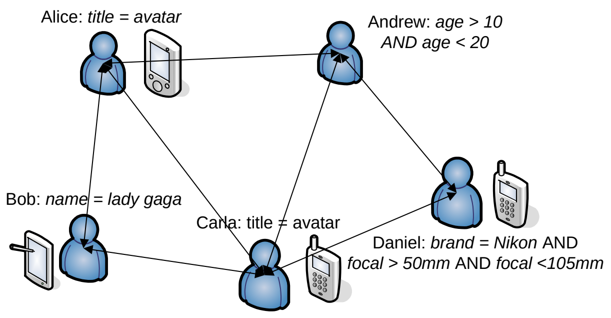

Alice: *title = avatar*
Andrew: *age > 10 AND age < 20*
Bob: *name = lady gaga*
Carla: title = avatar
Daniel: *brand = Nikon* AND *focal > 50mm* AND *focal <105mm*

Fig. 1. An example of HUNETs where each user is equipped with a mobile device. Their contact patterns among users are based on their social relationships. Each user has his/her own interests, which is represented as a *name:interest* pair.

Users in a HUNET can communicate with each other only when in vicinity. As a result, physical layer end-to-end connectivity is not available, and traditional connection-oriented networking technologies cannot work, or at least cannot work as efficiently. Additionally, it is noted in [14], [18] that users are becoming more and more interested in *contents* instead of connections. In other words, users care most about the content of the messages they received, not where they received them from. Therefore, the real problem in HUNETs becomes "*what* should to be sent to *who*?" i.e. *how to efficiently transmit messages to users that are interested in their contents?* A concrete example is *Twitter* [1], a popular social networking service. Twitter lets users "*follow*" users whom they are interested in. The "follow" mechanism grants users the ability to express their interests in the sense that they are generally interested in the contents generated by the people they are following. The fact that Twitter is extremely popular on wireless devices demonstrates that it is important for applications to have the capability of expressing users' interests. Additionally, providing such a function grants the users the ability to communicate without exposing their identities, which protects users' privacies.

Based on the above analysis, we propose to use *content-based publish/subscribe* [10] (CBPS or content-based pub/sub) in HUNETs. Fig. 1 illustrates the use of CBPS in a HUNET, where each user is interested in certain content specified by a *subscription* (represented as conjunctions of Boolean expressions). Users communicate with each other when they are within each other's communication range. Each user generates messages with labeled contents. Messages are forwarded to users who are interested in the content. This design eliminates the requirement of the end-to-end connection. It takes the advantage of the pervasive nature of HUNETs and provide a plausible application scenario.

In this paper, we present *ZigZag*, a comprehensive CBPS architecture featuring high efficiency and low complexity. Our motivation of designing a new architecture is that although existing pub/sub systems work well in the Internet, they do not consider the unique properties of HUNETs and will perform poorly. In designing ZigZag, we focus on two components: *content representation* and *pub/sub routing*. The paramount goal is to minimize its complexity, so to make it work on resource-constrained mobile devices. ZigZag uses *Z-order curve* to transform multi-dimensional ranges into scalars, and uses *Bloom filter* to compress transformed numbers. In this way, the complex multi-dimensional matching problem is reduced to a constant time query to the Bloom filter. Our contributions in this paper are as follows:

- We frame the concept of HUNET, a novel mobile application platform centered on content-based networking;
- We design ZigZag for HUNETs, a CBPS architecture based on transformational filling curve and Bloom filter;
- We perform extensive real-world-trace-based simulations to verify our design and analysis;

The rest of this paper is organized as follows: Section II presents related work. Sections III and IV give the detailed description of each component of ZigZag. Section V discusses the simulation results. Section VI concludes the paper.

II. Related work

1) Human-centric networking: DTNs [12] are similar to HUNETs, which share the same physical structure. The difference is that DTN research focuses on connection-oriented networking, whereas HUNET is more concerned with the efficient support of novel applications. Pocket network [8] shares the same model as ours, and also provides pub/sub-style services, but it does not explore the design of low-complexity protocols. DTN multicast algorithms [22] have been proposed. Although they can be used to support routing of pub/sub routing in the Internet, they do not work in HUNETs due to mobility. Our previous work [19], [23], [24] presented preliminary results on providing pub/sub communication in HUNETs.

2) Content-based pub/sub systems: Pub/sub is a powerful communication paradigm [10]. The use of pub/sub systems in mobile ad hoc networks has recently attracted interest [9], [20]. These methods do not work in HUNETs since they rely on end-to-end connectivity. Content-based networking has recently been used in opportunistic data diffusion [6]. In [7], the authors studied the content-based addressing and routing for large-scale pub/sub networks, which is too complex to be used in HUNETs. A recent work [21] uses content-based pub/sub systems for content dissemination in DTNs. Subscription representation and processing is a well-studied problem [11]. The transformational method has been used in MICS [15].

3) Network applications of the Bloom filter: Bloom filter [4] enables a trade-off between space complexity and query accuracy, which is proven to be useful in many network applications [5]. The use of Bloom filter in the pub/sub systems and the Internet can be found in various literatures [16], [17]. In [16], Bloom filter is used to encode efficient interest predicates. In [17], Bloom filter is used to encode the addresses of multicast destinations.

III. Transformation-based subscription representation and processing

A. Z-order-based subscription representation and processing

The standard way to represent a subscription is to use the conjunction form of multiple *attribute constraints*. An attribute constraint is an interval of an attribute. For example, Daniel's interest is represented by two attribute constraints $\{brand = Nikon$ AND $focal \geq 50mm$ AND $focal \leq 105mm\}$. Here, $brand$ and $focal$ are two attributes. The attribute constraint $brand = Nikon$ indicates that the user is interested in the messages of Nikon-branded lens. A subscription is a hyper-rectangle on the multi-dimensional content space. On the other hand, a message's content is represented by multiple numbers on each dimension, which is a hyper-point. A message matches a subscription if it falls inside the subscription's scope. For example, a message labeled with $\{brand = Nikon$ AND $focal = 85mm\}$ will be forwarded to Daniel.

There are various types of data structures for the storage and processing of large numbers of subscriptions [11]. These techniques are not applicable in HUNETs, for they require complex operations. Our approach, however, uses *approximate matching*, which is based on *transformation filling curve*.

The idea is to discretized the entire content space into cells, and use cells to approximate an arbitrary range in the multi-dimensional space. For example, in the left side of Fig. 2 shows that a 2D plane is divided into 16 grids. As shown in the right side of Fig. 2, the heavy-shaded rectangle is a subscription. It is represented by 4 cells intersected with it, which are light-shaded. In this way, a multi-dimensional range is transformed into multiple discretized cells.

Then, each cell is labeled with a number, so that a multi-dimensional range is approximated by a set of cell IDs. Cell IDs are assigned using filling curve. The filling curve we use is *Z-order curve*. A previous work, MICS [15], uses *Hilbert filling curve*. The benefits of Z-order is its lower computation complexity [2]. The right of Fig. 2 also presents the traversal sequence of cells using Z-order curve. Each cell's label is simply the sequence number of Z-curve traversing each cell. In the right side of Fig. 2, the ID of each cell is shown near its centroid.

To obtain the cell label, we do not need to really count the sequence number. For Z-order curve, a cell's ID is obtained by interleaving the bits of its coordinates' binary forms. For example, the coordinates of cell A in Fig. 2 is $[2, 2]$, or $[10, 10]$ in binary form. The corresponding ID after Z-order curve transformation is 12 or 1100 in binary form. It is clear that Z-order curve transformation can be computed efficiently using special vector instructions that are available in modern CPUs.

The set of cell IDs representing the original mD range is then compressed as intervals. A contiguous sequence of integers is represented by an interval whose two end-points are the first and last integers.

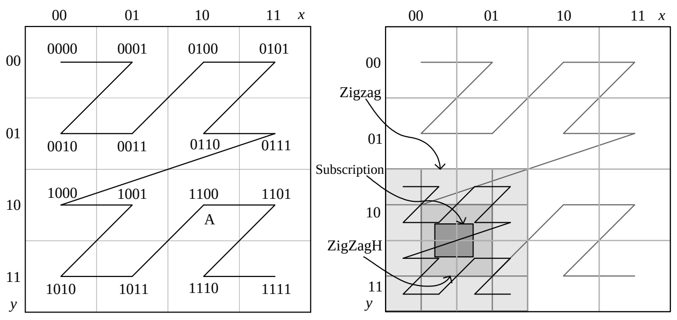


Fig. 2. An example of using Z-order curve in 2D plane. The plane is split into cells. Each cell is labeled with an integer number. The label of each cell is the sequence number of the Z-curve traversing all cells. It can be obtained by interleaving the coordinates of the cell in the original scale.

ZigZag transforms a subscription into multiple intervals. This transformation will introduce false matches in matching since the transformed range may not be identical to the original one. We have two strategies to determine how to perform the transformation with different false ratios:

- Inclusive transformation. Each subscription is represented by the IDs of the cells intersected with the subscription. This only results in false positives. It is expressed in the expression below:

$$FPR = \frac{\sum_{c_i \cap s \neq \emptyset} c_i}{s} - 1 \tag{1}$$

  In Eq. 1, s is the hyper-rectangle of the subscription, c_i is a cell of the content space.

- Optimal transformation. We use the byper-rectangle of cells that has the minimal difference from the subscription to approximate it. This results in false positives and false negatives. But, the false ratio would be no more than the above approach:

$$FPR = min\{\frac{\sum_{c_i \cap s \neq \emptyset} c_i}{s} - 1, 1 - \frac{\sum_{c_i \cap s = c_i} c_i}{s}\} \tag{2}$$

  The first term here is the same as in Eq. 1. The second one is all cells inside the hyper-rectangle of the subscription.

A false positive does not harm the users' satisfaction in the sense that matched messages will be delivered, but false negatives will. We argue that since our application is used for recreation, as long as the false negative is within a certain range, it does not harm the system's usability. In order to evaluate the benefits of introducing false negatives in ZigZag, we propose a metric called *relative beneficial factor* (RBF). It is calculated as follows: Suppose the false positive of the first approach is represented by FP, and the false negative, if exists, of the second approach is represented by FN. RBF is defined as the following equation:

$$RBF = \frac{FP}{FN} \tag{3}$$

Given a threshold α for the FN, and a threshold β for the RCF, we will employ the second approach only if:

- $FN < \alpha$
- $RBF > \beta$

These two conditions make sure that the false negative does not cause too much waste (the first constraint), and the benefits of tolerating false negatives pays off (the second constraint).

The transformed intervals of a subscription are inserted into an *interval tree* for fast querying of subscription matching. To check if an event matches a subscription, we first transform the event to a point using ZigZag, and then query the interval tree representing the subscription. If the point is in the tree, then the event matches the subscription; if not, the event does not match.

We can derive the FPR of transformation as follows: Suppose that the subscription space is a k-dimensional space, and the false positive rate at each dimension is p_i $(i \in \{0, ..., k-1\})$. The overall FPR is:

$$FPR = 1 - \prod_{i=0}^{k-1}(1 - p_i) \tag{4}$$

Since the FPR on each dimension is quite small, Eq. 4 can be approximated as $FPR = \sum_{i=1}^{k} p_i$.

B. Hierarchical ZigZag

Even we can minimize the false rate in matching using the conditions proposed in the previous section, but it still causes considerable false positives in extreme cases. For example, in the right side of Fig. 2, the size of the transformed subscription is much larger than the actual size.

We propose *hierarchical ZigZag* (ZigZagH) to further reduce the false positive rate. The ZigZagH refines the transformation by applying a finer grained transformation on the obtained results of the initial transformation. As shown in Fig. 2, a finer-grained ZigZag transformation is applied on the obtained subscription space. The false positive is significantly reduced. Here, the indexing method will be the same for the second level ZigZag, but its granularity is determined by the precision requirement of the application. ZigZagH needs two interval trees to store the coarse- and fine-grained intervals.

ZigZagH's matching is similar to ZigZag. The difference is that we need to do it twice to match a single subscription. We need two sets to represent the two-level representations of the subscription. An event matches a subscription only if it matches the both sets.

C. Subscription compression using the Bloom filter

Using Z-curve, ZigZag transforms subscriptions into intervals and stores them into an interval tree. The query speed is $O(logN)$, where N is the stored intervals. Since the scale of HUNETs is small, the occupied space of subscriptions is quite small compared to the entire content space. These tiny subscriptions scatter sparsely across the huge content space, and form a large number of intervals that comprise small number interval points. In Table I we give the average number of points contained in each interval for different dimensionality. Each dimension is separated into $2^{10} = 1,024$ cells. Interestingly enough, the value does not change too much.

Dimensionality	2D	3D	4D
Interval size	6.18	6.25	6.52

TABLE I
THE AVERAGE NUMBER OF POINTS CONTAINED IN EACH INTERVAL AFTER Z-ORDER TRANSFORMATION.

According to this observation, we use Bloom filter [4] to accelerate matching speed using the same amount of memory of interval tree. For the completeness of the discussion, we give a brief introduction to Bloom filter. Please refer to [4] for a detailed discussion about the data structure. Bloom filter is a data structure that represents set, which supports probabilistic membership querying. An empty Bloom filter is a bit-vector of m 0 bit. Bloom filter uses k different hash functions. To insert an element, it is fed each of the k hash functions to get k array positions. Set the bits at all these positions to 1. To test the membership of an element, all its hashed positions in the bit array must be "1". The FPR of a Bloom filter of m bits, using k hash functions, and storing n elements, is given in Eq. 5.

$$f = (1-(1-\frac{1}{m})^{nk})^k \approx (1-e^{\frac{-nk}{m}})^k \qquad m \gg n,k \quad (5)$$

The benefits of using the Bloom filter is its succinct storage and efficient processing. However, it results into additional FPR, besides that of Z-order transformation. Suppose that the FPR of transformation is P_T, and the FPR of using the BF is P_B. The FPR of combining these two techniques will be $1-(1-P_T)\times(1-P_B)$, which is equal to $P_T+P_B-P_T\times P_B$. Usually, the FPRs are quite small, so the combined FPR can be written as $P_T + P_B$. That is, using Bloom filter increases the FPR of subscription matching by its own FPR. But it only introduces a small amount of FPR with our settings. For example, suppose that $32bit$ integers are used in the transformation, each interval uses (at least) $64bit$ memory to store two end-points. As indicated in Table I, each interval contains about 6 integers. If we put 6 integers into a $64bit$ Bloom filter and use 3 hash functions, according to the FPR of Bloom filter [4], its FPR is only 1.35%, as calculated using Eq. 5. This is acceptable given that the FPR of Z-order curve transformation is in the same order.

IV. PUB/SUB ROUTING FOR HUNETs

Although multicast is used in many existing pub/sub routing protocols as the underlying forwarding technique, it may not be a good choice in HUNETs for the following reasons:

- It breaks user privacy. Using multicast, we have to know the identity of the destinations. This compromises users' privacy, and contradicts with the original intention of using pub/sub in HUNETs.
- It has high complexity. In [13], the authors analyzed how to use the community structure to enable multicast. However, there is no efficient method to collect such information beforehand. Even such information is available, the high complexity makes it inapplicable.

Algorithm 1 Message forwarding from broker B_i to B_j

```
1:  $B_i$ collects subscriptions from $B_j$;
2:  for all $sub_j$ from $B_j$ and $sub_i$ from $B_i$ do
3:      if $sub_j.TTL > sub_i.TTL$ then
4:          $sub_{ij} \leftarrow sub_i \wedge sub_j$;            /*bitwise and*/
5:          $B_i$ queries each message $msg$ against $sub_{ij}$;
6:          if $msg \in sub_{ij}$ then
7:              $msg.t \leftarrow sub_j.TTL - sub_i.TTL$;
8:          end if
9:      end if
10:     Sort all the messages according to the descending order
        of $msg.t$;
11:     $B_i$ forwards the top-ranked messages to $B_j$;
12: end for
```

ZigZag's pub/sub routing does not enforce rigid association between clients and brokers. Forwarding decisions are made on-the-fly when users meet. We also propose a method for users to compress multiple subscriptions received from others. We propose to make routing/forwarding decisions on-the-fly when two users meet. Users exchange routing information within the short contact duration. Logically, there are two operations: message collection from publisher to broker and message forwarding between brokers. For their small memory consumption and high efficiency, the subscription representation and processing methods we presented before are well suited for this purpose. Another enabling condition is the small scale of HUNETs, therefore the data needed to perform routing is fairly small.

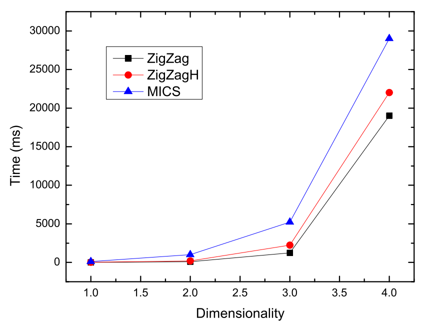
The forwarding algorithm is illustrated in Algorithm 1. Our design incorporates routing information in subscriptions. Our approach is simple: attach a time-to-live (TTL) value to each subscription sent out by users. A subscription will be removed from memory if its TTL expires. By limiting the cope of propagation, each user specifies the messages he/she is interested in in a limited time window. Each time a new subscription is collected by a user, its TTL is set to the value of the newly added value. So, a longer TTL value indicates a "fresher" subscription. As shown in Algo. 1, TTLs guide the forwarding of messages to users that have fresher TTLs and finally reach the destinations, which are interested in the content represented by the subscription.

V. PERFORMANCE EVALUATION

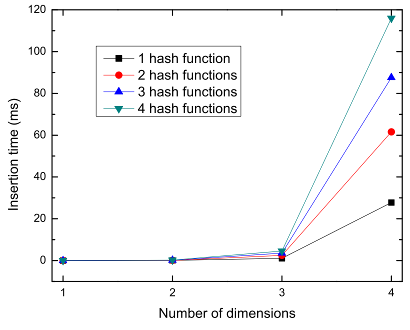
A. Processing speed

We implement all the methods using C++. We do not apply any optimization in the compilation. The program is executed on a Windows desktop computer. The machine has an Intel quad-core CPU (each core runs at 3GHz) and 6GB DDR3 memory. We do not use any parallelism in programming or compilation.

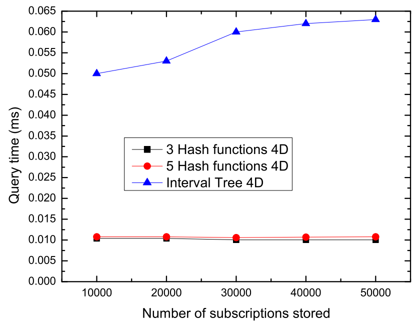
We first compare the processing speed of ZigZag with the Hilbert filling curve method of MICS [15]. Fig. 3(a) depicts the insertion time of a subscription, including the time used to transform a subscription into intervals and that of inserting



(a) Insertion time of interval tree



(b) Insertion time of Bloom filter



(c) Query time

Fig. 3. The transformation time, insertion time, and query time of ZigZag.

intervals into an interval tree. We prepare 100 subscriptions and then get the average value. In this set of experiments, each dimension is split into 1,024 intervals. And each subscription occupies $[0, \frac{1}{100}]$ of the entire range on each dimension, which is drawn uniformly.

ZigZag's insertion speed is slightly faster than MICS since the transformation using Z-order curve is simpler than Hilbert curve. ZigZagH's insertion time is longer than ZigZag. This is because ZigZagH performs two transformations. Where the overall time does not double compared to ZigZag, since the number of generated intervals in the second level transformation is less than the first level. We can see from the figure that the transformation time increases exponentially with the number of dimensions used in subscriptions. So, we should avoid using more than, say, 4 attributes. Although the performance listed here is quite pessimistic, it should be noted that there is not any optimization applied in either programming or compilation. And the optimization methods proposed in [15], including *subsumption, covering, and merging*, are not applied in this simulation. Thus, these results show only relative performance of different methods and should not be used as design guidelines.

Fig. 3(b) presents the average time used in inserting a transformed subscription into a Bloom filter. The insertion time of Bloom filter is much smaller than the interval tree. Since the transformed points of a subscription grows exponentially with the dimensionality, as long as the dimensionality is low, the insertion time is much better than the interval tree. Fig. 3(c) presents the average time used to query a single event against 100 subscriptions. The content space has 4 attributes. The above line represents the interval tree, which is also used in MICS [15]. The bottom two lines represent the results of using Bloom filter (3 and 5 hash functions). It is clear that Bloom filter runs much faster since it takes constant time to check the membership of the event in the subscription.

B. Storage and false positives

The memory consumption of the interval tree grows in $O(n)$, where n is the number of stored intervals. We present the average number of intervals after transformation for different dimensionality in Fig. 4. Note that the figure is in $log - log$ scale. The number of intervals grows cubically with the number of cells on each dimension, which is obtained from numeric fitting. Assuming that there will be less than 100 users in any HUNETs, which is reasonable considering real application scenarios, the total memory of storing 100 subscriptions for all users will be acceptable. MICS's hilbert filling curve actually has a slightly less amount of intervals due to its better locality. Locality brings more benefits when subscriptions span larger ranges. But in HUNETs subscriptions are sparse and small.

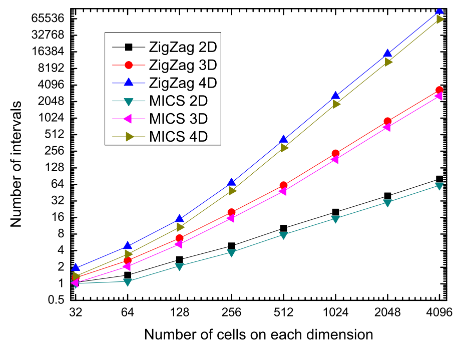


Fig. 4. The average number of intervals of a subscription after transformation versus the number of cells on each dimension.

The FPR of a query to a ZigZag subscription is shwon in Fig. 5. The FPR under coarse granularity of division of cells is extremely poor, and is unusable. The FPR drops to acceptable range beyond $2^8 = 256$ cells on each dimension. It is interesting to find that before that point, the FPR for higher dimensionality is much higher than that of lower ones. The reason is that high dimensionality causes more troubles when the false rate is high. The FPR drops below 3% beyond $2^{10} = 1,024$ cells. Because of the Bloom filter's hash-based property, the storage requirement of the elements does not affect the storage of the Bloom filters. That is, storing 10 $8bit$ numbers uses the same amount of memory as storing 10 $32bit$ numbers, and they have the same false positive rate. So, if we expand the range of the dimension on the subscription space, the storage used for Bloom filters does not need to change. The additional FPR of using Bloom filter is given in Fig. 6. The additional FPR is at most 1.6%.

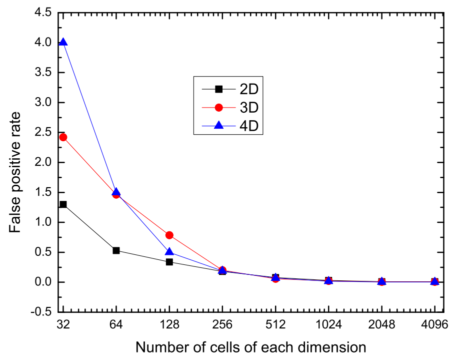


Fig. 5. Query FPR vs. the number of cells separated on each dimension.

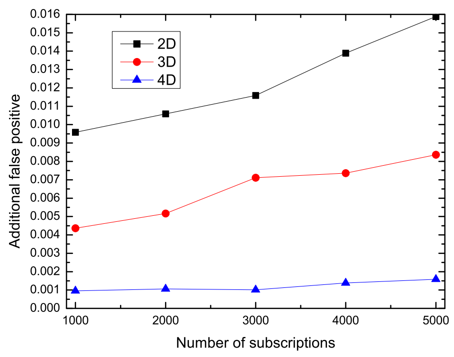


Fig. 6. Additional FPR of using Bloom filter. 4 hash functions are used. The bit-vector uses the same amount of memory as the interval tree.

VI. Conclusion

In this paper, we present a novel CBPS architecture called ZigZag for HUNETs. ZigZag employs a Z-order curve based subscription representation and processing technique, and a Bloom filter based message routing and forwarding protocol that are designed specifically for HUNETs. The combination of these techniques achieves high efficiency and low overhead. ZigZag demonstrates the benefits of novel architecture designs in a challenged network with scarce resources. We evaluate ZigZag using real-world-trace-based simulations. Our results verify that the performance of ZigZag is better than existing techniques. Our future work will be a prototype system working on smart phones.

Acknowledgement

This work is supported in part by NSF grants CCF 1028167, CNS 0948184, and CCF 0830289.

References

[1] "Twitter." [Online]. Available: http://www.twitter.com

[2] "Z-order Curve." [Online]. Available: http://en.wikipedia.org/wiki/Z-order_(curve)

[3] BBC., "New mobile message craze spreads," http://news.bbc.co.uk/2/hi/technology/3237755.stm.

[4] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. of ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[5] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," in *Internet Mathematics*, 2002, pp. 636–646.

[6] I. Carreras, D. P. Francesco, D. Miorandi, D. Tacconi, and I. Chlamtac, "Why neighborhood matters: interests-driven opportunistic data diffusion schemes," in *Proc. of CHANTS '08*. New York, NY, USA: ACM, 2008, pp. 81–88.

[7] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf, "Content-based addressing and routing: A general model and its application," 2000.

[8] A. Chaintreau, P. Hui, J. Crowcroft, C. Diot, R. Gass, and J. Scott, "Pocket switched networks: Real-world mobility and its consequences for opportunistic forwarding," *Selected Areas in Communications, IEEE Journal on*, vol. 26, no. 5, pp. 748–760, Feburary 2005.

[9] G. Cugola, A. L. Murphy, and G. P. Picco, "Content-based Publish-subscribe in a Mobile Environment," in *Mobile Middleware*, P. Bellavista and A. Corradi, Eds. Auerbach Publications, 2006, pp. 257–285, invited contribution.

[10] P. T. Eugster, P. A. Felber, R. Guerraoui, and A. M. Kermarrec, "The many faces of publish/subscribe," *ACM Comput. Surv.*, vol. 35, no. 2, pp. 114–131, 2003.

[11] F. Fabret, H. A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha, "Filtering algorithms and implementation for very fast publish/subscribe systems," in *Proc. SIGMOD '01*. New York, NY, USA: ACM, 2001, pp. 115–126.

[12] K. Fall, "A delay-tolerant network architecture for challenged internets," in *Proc. SIGCOMM '03*. New York, NY, USA: ACM, 2003, pp. 27–34.

[13] W. Gao, Q. Li, B. Zhao, and G. Cao, "Multicasting in delay tolerant networks: a social network perspective," in *Proc. of MobiHoc '09*. New York, NY, USA: ACM, May 2009, pp. 299–308.

[14] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. of CoNEXT '09*. New York, NY, USA: ACM, 2009, pp. 1–12.

[15] H. Jafarpour, S. Mehrotra, N. Venkatasubramanian, and M. Montanari, "Mics: an efficient content space representation model for publish/subscribe systems," in *Proc. of DEBS '09*. New York, NY, USA: ACM, 2009, pp. 1–12.

[16] Z. Jerzak and C. Fetzer, "Bloom filter based routing for content-based publish/subscribe," in *Proc. of DEBS '08*. New York, NY, USA: ACM, 2008, pp. 71–81.

[17] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "Lipsin: line speed publish/subscribe inter-networking," in *Proc. of SIGCOMM '09*.

[18] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet inter-domain traffic," in *Proc. of the ACM SIGCOMM'10*. New York, NY, USA: ACM, 2010, pp. 75–86.

[19] F. Li and J. Wu, "MOPS: Providing content-based service in disruption-tolerant networks," in *Proc. of ICDCS '09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 526–533.

[20] T. Pongthawornkamol, K. Nahrstedt, and G. Wang, "The analysis of publish/subscribe systems over mobile wireless ad hoc networks," in *Proc. of ACM MobiQuitous '07*, Aug. 2007, pp. 1–8.

[21] G. Sollazzo, M. Musolesi, and C. Mascolo, "TACO-DTN: a time-aware content-based dissemination system for delay tolerant networks," in *Proc. of ACM MobiOpp '07*. New York, NY, USA: ACM, 2007, pp. 83–90.

[22] W. Zhao, M. Ammar, and E. Zegura, "Multicasting in delay tolerant networks: semantic models and routing algorithms," in *WDTN '05: Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. New York, NY, USA: ACM, 2005, pp. 268–275.

[23] Y. Zhao and J. Wu, "B-sub: A practical bloom-filter-based publish-subscribe system for human networks," *Distributed Computing Systems, International Conference on*, vol. 0, pp. 634–643, 2010.

[24] ——, "Socially-aware publish/subscribe system for human networks," in *2010 IEEE Wireless Communication and Networking Conference*. IEEE, April 2010, pp. 1–6.