Locus: A Location-based Data Overlay for Disruption-tolerant Networks*

Nathanael Thompson, Riccardo Crepaldi and Robin Kravets
Department of Computer Science, University of Illinois Urbana-Champaign
201 N. Goodwin Ave.
Urbana, IL 61801-2302
{nathomps,rcrepal2,rhk}@cs.illinois.edu

ABSTRACT

Embedded sensors in mobile devices such as cars and smart phones present new opportunities to collect location-specific data about an environment. This data can be used to enable new real-time location-based applications. A major challenge is efficiently collecting, storing and sharing the data. This paper proposes Locus, a location-based data overlay for DTNs. Locus keeps objects at specific physical locations in the network using whatever devices currently are nearby. Nodes copy objects between themselves to maintain the locality of data. Location utility functions prioritize objects for replication and enable location-based forwarding of data look-ups. As a first-of-its-kind application, Locus is compared against other possible replication policies and shown to achieve query success rates nearly 4 times higher than other approaches.

Categories and Subject Descriptors

C.2 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Performance

Keywords

DTN, Data Overlay, Location-based

1. INTRODUCTION

Computer networking is on the threshold of a shift in prioritization. A near-term future can be envisioned in which most vehicles and smart phones are equipped with an array of various sensors collecting data about the surrounding environment. The broad deployment of embedded sensors will lead to wide-spread participatory sensing by end-users and enable the generation of unprecedented amounts of data. User demand for access to these large volumes of data is creating a shift in focus towards content- and data-centric systems and away from traditional end-to-end communication [8]. Given the participatory nature of such environmental sensing, data is inherently tied to the location where it was generated and not to the user who created it. In these new environments, the primary entities of the network will be data objects rather than individual devices. In a mobile environment with limited resources, the accumulation of such location-specific data raises two important questions of where to store the data and how to access it.

At first glance, remote access, where data lives on centralized servers (e.g., Cartel [6]), seems to be a natural choice. Using a mobile connection such as 3G or WiMax, the collected information can be uploaded to central storage and retrieved by other nodes from the same storage. However, with the increasing number of devices generating data and the high rates at which data is generated, the bandwidth requirements will quickly overwhelm the infrastructure, especially considering that those networks are already pushed to the limit to serve existing mobile Internet access.

Instead, given the location-centric nature of the data, a better alternative approach is to store data on the mobile devices themselves. This reduces load on the congested wide-area infrastructure and also keeping the data near the nodes that generate and consume it. Many distributed data overlays have been developed for the Internet that store data on end-nodes (e.g. Chord [15]). However, current overlay solutions are inherently node-focused, mapping data objects to specific nodes in the network. In vehicular and other mobile networks where nodes have high speeds of movement and quickly gain and lose connectivity, the environment is too unstable to manage such mappings without excessive overhead. Additionally, the intermittent connectivity in these networks [3, 13, 21] renders most existing solutions ineffective due to restrictive assumptions about network connectivity. To overcome the challenges of these intermittently connected environments, what is needed is a data overlay that can run on top of a disruption-tolerant network (DTN).

The location-based nature of the collected data, where data is inherently tied to a specific location or region, points towards caching of data in the geographic area where it was generated. Rather than moving objects to particular nodes in the network, the data objects are stored on the devices available in the data's home geographic area, tightly coupling data to the physical location it describes. To find data, a node sends a request to the target location instead

*This research was sponsored in part by National Science Foundation Grant 0081308.

CHANTS'10, September 24, 2010, Chicago, Illinois, USA.

of searching for a specific node or data id. The goal of such a system is to keep each data object as close to its home location as possible, given the currently available storage options in that area. The challenge to enabling a location-based data overlay comes from the high mobility in the network. Since devices are always on the move, the set of nodes near an object's home location is in constant fluctuation.

The main contribution of our paper is design of Locus, a location-centric data overlay for disruption-tolerant networks that stores data at physical locations and provides data look-up through location-based forwarding. The novelty of our approach comes from decoupling the data from the nodes that carry it. Essentially, data in Locus is explicitly tied to the location where it was collected. As devices leave and enter different areas, they exchange data with passing nodes using the location information available to them to filter and order the set of messages replicated. Essentially, nodes pass stored data to other nodes that are in a better position for that data and receive data that lives near their future location. Support for this dynamic caching is enabled using replication-based techniques similar to message forwarding protocols in disruption-tolerant networks (DTNs) [1, 14]. Unlike existing approaches, however, in Locus, replication has been adapted to support data storage and location-based query/response rather than generalized end-to-end connectivity.

To manage the limited bandwidth available during encounters between mobile nodes, Locus uses a utility-based message ordering scheme for forwarding and dropping data objects based on the data's home location and the node's current location. Combined with these ordering functions, alternate utility functions for query and response forwarding serve to balance extending data availability and increasing query match rate and response delivery. Simulation results in networks of moving vehicles show that Locus's policies are effective at keeping data near its home location. In turn, Locus's storage techniques, when coupled with location-based forwarding, lead to high query match rates and response delivery rates, achieving nearly 4 times higher query success rates compared to regular SCF forwarding approaches.

In the next Section, the shortcomings of existing data management approaches in the environment that Locus targets are described. Section 3 outlines the Locus architecture and provides details about the data storage components. Section 4 covers Locus's data access components. Simulation results are presented in Section 5, and the paper finishes with a conclusion and future work in Section 6.

2. DATA MANAGEMENT IN DTNS

Location-specific data is the enabling component of a new class of applications. Access to near real-time and recent historical data about a specific location can be used to drive applications such as optimized vehicle route planning, real-time fuel efficiency and location-based data sharing such as advertisements, announcements and media sharing. All of these applications need fine-grained and up-to-date information about specific locations for the best operation, since parameters such as traffic and weather conditions can fluctuate quickly and have a large impact on application decisions.

The challenge comes from the fact that the devices that are collecting and accessing the data are mobile nodes, including vehicles and smart phones with local Wi-Fi or Bluetooth connectivity and possibly some form of wide-area wireless connectivity. The mobile nodes have some local storage and limited computational power. As the number of these devices equipped with sensors increases, the amount of data generated quickly increase. What is needed is an efficient system that can collect, store and publish these large volumes of data for use by the above applications.

2.1 Centralized Approaches

Many existing systems use a centralized or infrastructure-heavy approach for data management. Some approaches assume that all mobile nodes are one hop away from a relay that can provide access to servers on the Internet [6]. Other solutions have proposed multi-hop communication between the mobile device and the relay [6, 7, 22]. However, all of these solutions require either expensive deployment of new basestations and infrastructure or assume that data traffic can be carried by cellular networks.

While cellular networks at first glance offer an attractive backhaul medium, the reality is that the load generated from participatory sensing systems will overwhelm existing cellular infrastructure. For example, in Chicago there are on average 2500 vehicles per km^2. If only 10% of those vehicles are generating data, the network must be able to support 250 data streams per km^2. A typical 3G basestation coverage radius is several hundred meters, or approximately 100 users. Given an upload capacity of around 500Kbps per basestation, that leaves only 5Kbps per device. While this data rate can support simple sensor readings, transferring larger data objects like images or mp3s will exceed the capacity of the network. More importantly, whatever capacity the basestations do have is shared with competing applications for mobile Internet access. Given that the demand for mobile Internet access will only increase, cellular approaches for location-based applications likely will not have sufficient bandwidth.

2.2 Decentralized Approaches in Connected Networks

Decentralized in-network data overlays can store data and keep it off the already burdened infrastructure while still retaining high data availability and accessibility. However, using the mobile devices themselves to store and share data poses additional challenges. In a network of mobile devices, any system must cope with disrupted connectivity, limited bandwidth from short contact durations and limited storage capacity [3, 13, 21].

In-network storage has been proposed for sensor networks to support replica placement that improves data lifetime, reduces look-up delay and reduces network overhead [16, 20]. These approaches are designed to overcome device energy constraints but are not well-suited to handling the limited bandwidth, connectivity and storage in networks of highly mobile devices. In addition, existing systems typically assume single hop connectivity between data and interested nodes. On the other hand, distributed hashtables (DHTs) do not assume single-hop connectivity, providing both storage and look-up functionality on edge nodes using structured routing techniques based on object and device ids [15]. However, the network model underlying DHTs assumes stable connectivity between all nodes participating in the overlay. Any system that requires stable connectivity between nodes will have low probability of finding data in DTNs.

2.3 Intermittently-Connected Networks

To overcome the limited connectivity, bandwidth and storage in DTNs, DTN-aware solutions for storing and sharing data are needed. Previous research has addressed managing the limited resources in DTNs, but with a focus on enabling end-to-end communication between devices. However, the data management techniques of these protocols are destination-oriented [1, 9, 12, 17] and therefore not directly applicable to data storage where there is no specific destination. While some replication schemes only use message-based metrics such as message age or number of message copies made [4], and so are agnostic to the actual destination, these schemes are not effective at making data easily accessible for mobile nodes, as will be shown our evaluations.

Existing DTN routing protocols are not only inappropriate for data storage in the overlay because of their destination-based focus, existing protocols are also ill-suited for performing data look-up. To directly apply destination-based forwarding, there must be a mapping service available in the network that indicates which node holds which data object. Maintaining and accessing such a service will be unreliable due to the high dynamics in the network and the lack of stable end-to-end connectivity. Without knowledge about the location of the data object or the network state, single-copy routing [2, 9, 10, 19, 22] will be unreliable. While multi-copy forwarding has been developed to operate without high levels of network knowledge [1, 4, 12, 14, 18], it is designed to probabilistically find devices instead of data objects. Unless there is a synergy between the data storage and data look-up, the probability to find data objects will still be low.

The real problem with existing solutions is that data is mapped to nodes in a manner such that data objects could reside anywhere in the network. As nodes move, data moves with them and disrupted connectivity will make it difficult to find a particular piece of data. Content-centric networking [8] avoids the dependence of data objects on specific nodes, but assumes hierarchical connectivity in the network that does not exist in DTNs. Instead of attempting to improve node-based storage for DTNs, we present Locus, a location-centric data overlay for DTNS, which avoids the problem by mapping data to physical locations instead of specific nodes. In this way, it is always clear where data is stored in the network. The challenge lies instead in managing the limited resources in the network.

3. LOCUS DESIGN

To support data storage and access in DTNs, Locus implements a data overlay on mobile nodes using a simple approach that does not require global information about the network and can handle the limited resources of DTNs. The primary entities in Locus are data objects, which can contain any type of information, from a single sensor value to a composition of multiple pieces of information. Throughout the rest of this paper, we use the term data to refer to data objects.

As mobile devices move through the environment, they periodically create new data. When data is created, it is tagged with the current timestamp, an application-specific type indicator and the current location, which becomes the *home location* for the newly created data. This location-based design requires that each device is equipped with a GPS or other sensor to capture the device's current position. While Locus can operate with any location service, applications benefit from more accurate measurements that support the collection of fine-grained information.

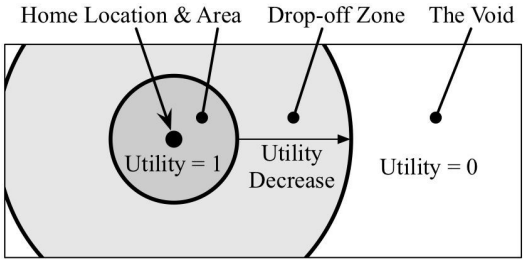


Figure 1: Storage bubble for a data object. Utility is highest at the center of the bubble. Utility drops off following a gradient around the edge of the bubble, creating a drop-off zone that pushes data back into the bubble.

The functionality of Locus is divided into two parts: data management and data access. The goal of data management in Locus is to maintain the storage of data at or near the data's home location. To this end, Locus provides novel virtual location-centric data storage that is managed through (i) intelligent replication management policies, which determine what data is forwarded to a node during a given contact opportunity, and (ii) intelligent cache replacement policies, which determine what data is removed to make space for new data.

The goal of replication is to maintain data as close as possible to the location where it was sensed so that the data can easily be found later on by queries. Instead of storing data on a particular node, Locus copies data to whatever nodes are currently near the data's home location. Due to node mobility, the set of nodes near the home location is always changing. To accommodate for the dynamic set of devices around a given location, Locus attempts to keep all data within a "bubble of knowledge" around the data's home location.

To access data in Locus, nodes forwards query messages to the physical regions in which the node is interested. Every query message contains a *target location* that is the location of interest and the type of data desired. As queries are forwarded toward the target area, any intermediary node can respond to the query if it contains any data whose home location matches the target location. Once matching data is found, a response message containing the data value is forwarded back through the network to the original requesting node.

The rest of this section gives the details of Locus's data management components including location-based utility, data replication policies and cache replacement policies. Locus's data access components are then described in in Section 4.

3.1 Data Storage

An individual data element d is defined to be the tuple $(\chi, \gamma, \tau, \phi, \zeta)$, where χ is the home location of d, γ is the size of the *home area* within which d should stay, τ is the time d was created, ϕ is the application data type and ζ is the value of the data (for instance, the temperature reading if the data is a sensed temperature data point). Since data

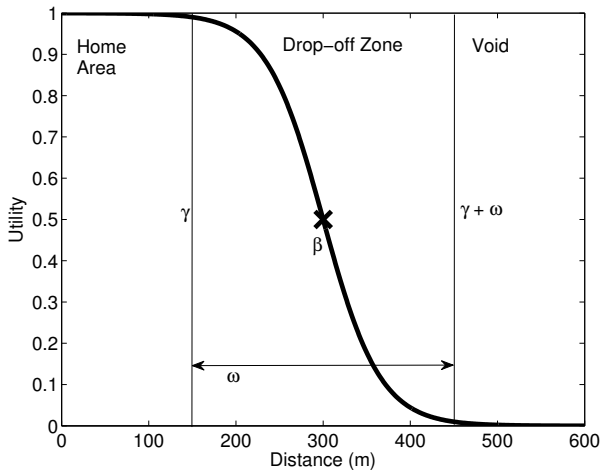


Figure 2: Equation 1 with $\gamma = 150$ and $\omega = 300$. This is generated from parameters $\beta = 300$ and $\alpha = 0.307$.

should be stored in the nodes near the data's home location, the probability of a node maintaining the a replica of the data diminishes the further the node travels from the data's home location. To enable this smart data caching, Locus uses location-centric utility functions. The basic idea is that in the home area surrounding the data's home location, the replicas should have a utility of 1 (see Figure 1). Around the home area is a drop-off zone where utility of the replica diminishes gradually, until it has a utility of 0. Essentially, the drop-off zone gives replicas a second chance to be pushed back into the home area.

To capture this behavior, Locus defines a utility function based on the distance δ from a specific location to a data replica d's home location $d.\chi$. The utility function maps $d.\gamma$ and δ into a value in $[0.0, 1.0]$. Locus's current utility functions are based on a parameterized sigmoid function:

$$u(\delta) = 1 - \frac{1}{\left(1 - e^{\alpha \cdot (\beta - \delta)}\right)} \quad (1)$$

In Equation 1, β is the point where utility is 0.5 and α determines the rate at which the utility changes. The shape of Equation 1 is shown in Figure 2 for $\alpha = 0.307$ and $\beta = 300$. As can be seen, the choices of α and β determine the home area size γ and drop-off zone size ω. γ is the point where utility starts to drop, roughly, $u(\gamma) = .99$. The drop-off zone extends from γ to the point where utility nears 0, so $u(\gamma + \omega) = .01$. α determines the size of ω by controlling how fast the function moves from .99 to .01. By solving Equation 1 with a fixed α and β for $u(\delta') = 0.99$ and $u(\delta'') = 0.01$, $\omega = \delta'' - \delta'$. Given ω, the value of γ is $\gamma = \beta - \omega/2$. By precomputing the different ω values resulting from specific settings of α, an appropriate α and β can be chosen for each data object depending on the application specifications.

The value γ plays an important role in the behavior of the system. If the home area is too small, it is possible that no nodes will be in the area and data will be quickly dropped. At the same time, if γ is too big, data will start to drift too far away from their home locations, making look-up difficult. At a minimum. γ should be large enough so that the home area always contains at least one node. Ultimately, the correct setting of γ depends on the density of the network at the specific location where d is created and the accuracy of location measurement available to nodes. Based on these parameters, γ should be set dynamically for each data. Currently, Locus uses a fixed γ. Dynamically adjusting γ based on network conditions is an important next step for our future work.

3.2 Data Replication

When two nodes meet, each node must select which data to replicate to the other node. Although each node will attempt to copy as much data as possible, because the contact duration is limited due to node mobility, the order of replication must be managed. Therefore, each node sorts data to replicate based on the data's utility to the system. Additionally, data is sorted by peer/data tuples and only replicated if the peer does not already have the data. Going in order of highest replication utility, the data from the first tuple is copied to the corresponding peer until either all data has been replicated or the contact ends. There are three possible cases when it is especially beneficial to replicate data.

First, if a node is currently in the data's home area, copying the data to surrounding nodes increases the probability of a replica remaining in the home area. In this case, data is always replicated to peers. Next, if the node is not in the home area, then it considers the location of the peer. If a node is in or near the drop-off zone, the data should be replicated to peers closer to the home location. Finally, even if a node and its peer are outside of the data drop-off zone, if the peer node is moving into the drop-off zone in the near future, the data should be replicated to the peer so that the data is carried back into its home area.

Formally, Equation 1 is applied to the distance $dist()$ between node position $n.pos$ and the data's home location $d.\chi$. The replication utility for each case is given as:

$$rep_1(d, n, p) = u(dist(n.pos_{current}, d.\chi)) \quad (2)$$

$$rep_2(d, n, p) = u(dist(p.pos_{current}, d.\chi)) \quad (3)$$

$$rep_3(d, n, p) = \epsilon \cdot u(dist(p.pos_{future}, d.\chi)) \quad (4)$$

The final replication utility is the maximum of the three functions:

$$rep(d, n, p) = max(rep_1(d, n, p), rep_2(d, n, p), rep_3(d, n, p)) \quad (5)$$

Each of Equation 3 through 4 gives a utility measure for replicating the data. When determining the final replication utility for data, we want to know what is the highest utility for this data if it is replicated to the peer. Since all three cases will be true after replicating, the highest utility is the max of Equation 3 through 4.

Estimating the future position of a node can be done simply by storing a small history of previous locations and computing the current device trajectory. When nodes meet, they exchange their current trajectory. While simple, this method is prone to prediction errors and privacy concerns. To improve accuracy, other prediction mechanisms could be used like the current driving directions or final destination in the GPS unit. To account for varying degrees of error in prediction, Equation 4 is scaled by a factor $\epsilon \in [0.0, 1.0]$ that can be adjusted depending on the prediction method used. In the case that users are unwilling to share their future location for privacy reasons, ϵ is set to 0 to suppress using future knowledge.

3.3 Cache Replacement

Locus stores data as long as possible to increase the probability of matching queries. However, when storage space is completely consumed, some data must be dropped. To select data to be dropped, node n finds the data d in the local buffer with the lowest utility, which is the data that is farthest away from its home location, i.e. Equation 2 with $p = null$ is lowest, and for which n is not moving into the data home area, i.e. the future utility of n is also low:

$$drop(d, n, p) = \epsilon \cdot u(dist(n.pos_{future}, d.\chi)) \quad (6)$$

Data with the lowest maximum utility between Equations 2 and 6 are dropped first. If the utility of two data replicas is the same because both data objects were generated at the same location or both are out of their drop-off zone, then the creation time $d.\tau$ is used as a tie-breaker and older data is dropped first.

4. DATA ACCESS

Data access in Locus is achieved through query and response messages. Query messages are passed between nodes until they reach a node with matching data. Whenever a query message matches some data, a response message is generated that is passed back to the original node. Each query message contains several parameters that specify how the query should match data. The primary matching constraint is location. A query message is always generated for a specific *target location* to find data about that location. Included in the message is also a range that defines the *target area*, similar to the relationship between the home location and the home area for the original data. According to Locus's location-based data storage, if the query is forwarded toward the target location, it should find the data that was created at that location. In addition to matching location, queries also can match data based on creation time and data type.

When a query message finds matching data, the node generates a response message containing the value of the matching data, $d.\zeta$, and the id of the query that was matched. Response messages are forwarded back to the node that originated the query in the same way that queries are forwarded. The target location of a response message is the location of the requesting node. Defining the target area is a challenging step for response messages. Nodes are constantly moving and a node's current location is difficult to know, making it difficult to set the response message target location. To overcome this challenge, Locus does several things. First, the querying node includes its current location in the query message when the message is first generated. Second, the node also includes its current trajectory in the query message. Third, when a node generates a response message for a particular query, it uses the enquirer's original location, trajectory and the age of the query to estimate the querying node's current location. Based on the age of the query, the target range is adjusted to account for increasing error in estimating the target location. If the future location is not available, the target range must be very large.

The rest of the section discusses Locus's specific mechanisms for forwarding query and response messages. The term "messages" will be used to refer to both query and response messages since the forwarding behavior is the same for both.

4.1 Message Forwarding

The basic structure of messages is similar to data. Each message has a target location χ and also a variable range ρ that defines the target area, similar to the data home location and area defined by $d.\chi$ and γ. In addition to location information, a query also contains time, type and node parameters. To match creation times, queries specify a valid time range. τ_{start} is the oldest acceptable creation time of data and τ_{end} is the newest acceptable creation time for data. Either or both of τ_{start}, τ_{end} can be set to infinite to indicate an unbounded time. If both are infinite, then there is no time constraint for matching objects. ϕ is the type of data the query wants to match and η is the information about the requesting node including location and trajectory. A query is the tuple of all these parameters $q = (\chi, \rho, \tau_{start}, \tau_{end}, \phi, \eta)$. Response messages contain a target location and range, but only include the matched data value ζ, $r = (\chi, \rho, \zeta)$

Because of the similarity between objects and messages, it seems natural to calculate a message m's forwarding utility by applying the same utility function used to replicate data, using $m.\chi$ and $m.\rho$ instead of $d.\chi$ and γ. Indeed, once a message reaches the target location ($rep(q, n, p) > 0$), Equation 5 can be used to find message forwarding utility and get the message into the target area. However, in most cases, queries will be generated far from the target location. If Equation 1 is used, the utility for any message outside of the target area is 0 or close to it. If Equation 5 is used to calculate the message forwarding utility, those messages will have the lowest utility and never be forwarded.

To move messages through the void where location utility is 0 (see Figure 2), a new utility function is needed for forwarding. The aim of Locus's forwarding is to move messages as quickly as possible toward their target location. At each encounter, the message should be given to the node that can carry it closest to the target location. Existing approaches use complicated models that require deep knowledge of the map on which devices are moving, the traffic levels on different streets and the destinations nodes are heading towards [19, 22]. Locus uses a more probabilistic approach that is better suited for DTNs where some nodes do not move on predictable maps and which integrates better with the replicating of data objects.

To decide if a message should be passed to a different node and if so, to which node to forward it, nodes compare the future location of connected peers to their own. The message is forwarded to whatever node has a future location closest to the target location. To ensure that these messages have a high enough utility to compete for bandwidth, if Equation 5 is 0 for a given message, the message is given a large utility value, $\Upsilon < 1.0$. In this way, messages that are far from the target location will be more likely be forwarded and quickly move towards the location.

The final challenge in forwarding messages comes from the fact that messages share the same limited contact opportunity as data objects. At each encounter, a node must not only decide what data to replicate, but also which messages to forward and in which order to send everything. To do so, appropriate utilities must be assigned for messages that balance the use of contact opportunities between data and messages. If messages are given too high utility, the transfer of messages will consume all of the bandwidth and prevent data from being replicated to their home area. On the other

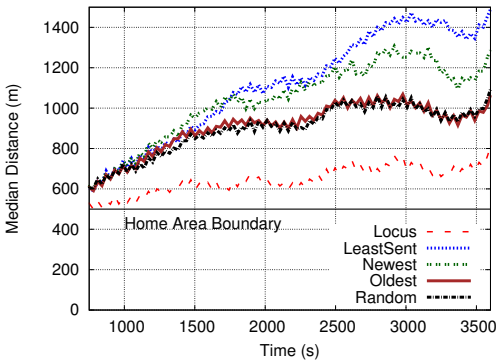


Figure 3: The change over time in median distance of all objects from their home location.

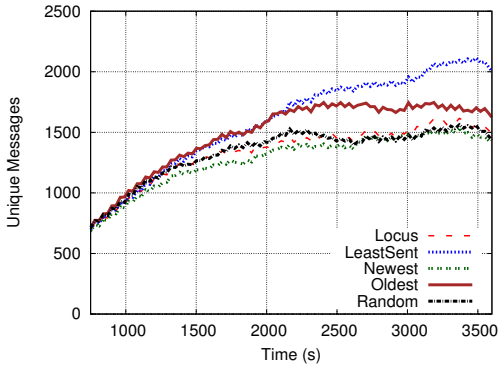


Figure 4: The change over time in number of unique messages in the network.

hand, if message forwarding is too passive, messages will be starved and look-up success rate will be low.

To achieve a balance between forwarding messages and replicating data, Υ is large enough that messages have high utility, but not too high to starve out data objects. In this work, $\Upsilon = 0.95$. To select the best peer to take the message m, the utility for a message is scaled for each peer p according to the peer's distance by the value $1/dist(p.pos_{future}, m.\chi)$. The message is forwarded to the peer for which the utility is highest, or kept at the local node if it has the highest utility. The forwarding utility for message m at node n to peer p is:

$$forward(m, n, p) = \Upsilon + \frac{1}{dist(p.pos_{future}, m.\chi)} \quad (7)$$

4.2 Multi-copy Forwarding

The main goal in Locus is to satisfying data look-ups, which require a round trip communication between a node and a location. For a look-up to be successful, two messages must be delivered, the initial query and corresponding response. While single-copy message forwarding limits the system-wide overhead, it achieves lower delivery probability in unpredictable environments [14]. If single copy routing were use, the low delivery probability would be compounded by Locus's roundtrip communication style. On the other hand, multi-copy approaches can exponentially increase the message load since for each copy of a query message that matches data, multiple response copies might be generated. This effect is especially bad in Locus since response messages contain the data itself and can be large. Ultimately, the requesting node must balance the message load and the expected delivery probability.

To achieve this balance, Locus uses a quota-based multi-copy forwarding approach [12, 14] for query messages and response messages are forwarded using single-copy location-based routing. When a query copy matches data, a response is sent and that copy is deleted. In this way, the explosion of response messages is avoided, but multiple queries and responses will still be generated for each look-up. The load on the system is balanced by the size of the initial quota allocated for each message. Response messages always have initial quota 1. To incorporate quota-based forwarding of queries with the replication utilities used by Locus, a new utility function is needed. Given an initial quota, $initialQuota$, the utility for a message m with remaining quota $m.quota$ is

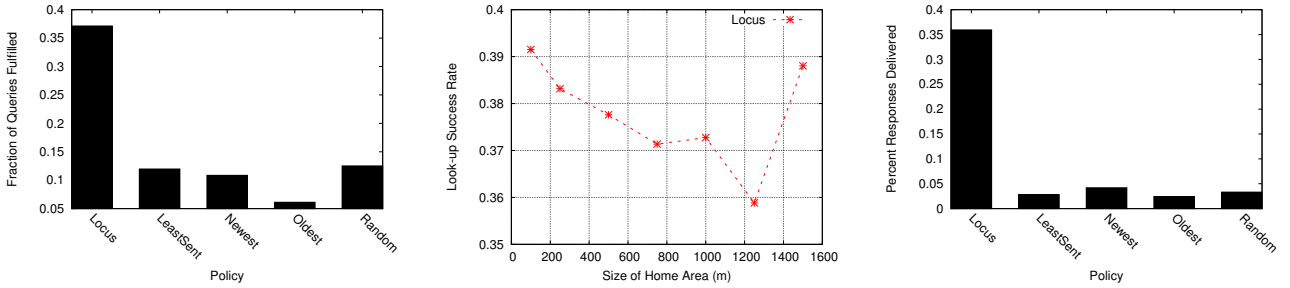
$$forward_{quota}(m, n, p) = m.quota/initialQuota \quad (8)$$

The final utility for a query message is the max of Equation 7 and Equation 8.

5. SIMULATION EVALUATION

The goal of the evaluation is to show that through its location-based policies Locus can i) keep data in its home area and ii) that having data close to a known location improves the probability of queries finding the data. Locus is evaluated in a simulated environment of instrumented vehicles. This environment presents physical distances and device speeds similar to what is expected in real deployments. However, due to the limitation of simulation, it is difficult to create scenarios with the node density similar to what is expected in real life. In the simulation scenarios, 150 vehicles move in a 5km x 5km area following movement over a map of Chicago generated using the VanetMobiSim [5] vehicle simulator. While this density of devices is far below that expected in real deployments, the results obtained in this environment will actually be far worse than in a real deployment because of the sparseness of the network. To ensure that home areas are covered by at least one node, the γ is set larger than normal with $\alpha = 0.307$ and $\beta = 650$ resulting in a drop-off zone of 300m and home area size of 500m. The drawback is that the number of individual data objects each node has to store is higher than normal. The traces were used to control mobility in the ONE simulator [11] to simulate data traffic.

The second challenge for evaluating Locus is to find a suitable comparison protocol. Existing SCF forwarding protocols cannot be directly applied for storing and replicating data because they are destination-focused for end-to-end forwarding of messages rather then persistent storage. To compare Locus against other possible overlay implementations, different message-based replication policies from DTN environments are used to store data and combined with basic Epidemic forwarding [18]. The different replication policies affect the way data is stored in the network and queries and responses are flooded to maximize the search. Several replication strategies are taken from literature, including LeastSent, which always replicates the data that has been copied the least, Oldest, which always replicates the oldest data,



(a) Look-up Success Rate (b) Varying Size of Home Area (c) Delivery Rate of Response Messages

Figure 5: ((a)) Overall look-up success rate using different policies to satisfy queries. ((b)) Change in query success rate using Locus as the size of home area changes. ((c)) Locus achieves higher query satisfaction rate through improved message forwarding, seen here in percentage of response messages actually delivered.

Newest, which replicates the newest data first and Random, which randomly replicates data.

In each scenario, every node generates new 1KB data objects every 10 seconds, mimicking a sensor application that profiles traffic or environmental conditions. Each node has a buffer big enough to hold the data generated from three nodes over the length of the simulation. Transmission rate is 2Mbps and nodes move along streets at speeds between 10mph and 55mph.

5.1 Data Storage

The effectiveness of Locus's location-based data replication policy is evaluated by comparing it against the other replication policies. The goal of Locus's replication policy is to keep data as close to the home location as possible to support location-targeted queries. Because Locus prioritizes data for replication based on its distance from the home location, the Locus policy should be significantly better at keeping data within the home area. Other replication policies do not take distance into account and therefore allow data to drift throughout the network. This fact is illustrated in Figure 3, which shows the change over time in the median distance of each data object to its home location. Locus's prioritization functions successfully keep data near or within its home area.

The trade-off in Locus's replication policy is that it does not focus on preserving data lifetime. The result is that some data is dropped from the system earlier than with other policies. Figure 4 shows that Locus tends to make more copies of fewer data objects in order to keep the data close to the home location. On the other hand, LeastSent attempts to ensure that all data has an equal number of copies in the network. The sacrifice is that data is spread throughout the network and is not necessarily close to the home location.

5.2 Data Access

The replication policy is useful only in how well it can preserve data to be found by application queries. To evaluate the query satisfaction rate using different policies, the query satisfaction rate using different replication policies was measured. Queries were generated by randomly selected nodes at a fixed rate network-wide. For each new query the target location was selected randomly from a location on a street in the space. This ensures that it is possible for data to have been generated in range of the target location. The query range was set to 200m.

In the first experiment, the overall query success rate was measured. Each query was given an infinite time range to focus on the interaction between message forwarding and data copying policies. As can be seen in Figure 5(a), Locus's location-based policies were far more effective at connecting users to data than other policies. This is a combination of Locus's policies of keeping data in the home area and the location-augmented forwarding techniques. Despite the high number of unique messages preserved in the system by LeastSent and Oldest policies, the messages are scattered around the network and difficult to find, even using Epidemic-style flooding, resulting in lower query success rate.

The performance of Locus depends on the size of the home area. Figure 5(b) shows how the balance of replicating data versus forwarding messages affects success rate. At small home area size, there is little replication and more bandwidth to forward messages. When the home area size is large, data is spread out across more nodes in the network allowing queries to more quickly find matching data. These results highlight the importance of the home area size.

The benefits of Locus come from a synergy between the location oriented data replication and the location-augment message forwarding. To demonstrate the importance of the message forwarding component, the effectiveness of delivering responses after a query was matched was measured. Because of the node mobility, this is a difficult but important phase of the entire process. As can be seen from Figure 5(c), a big part of Locus's improved query response rate comes from its unique response forwarding policies. By focusing a message to a specific area before spreading it, Locus can greatly increase the probability of finding the original node.

However, as illustrated in Figure 4 Locus sacrifices data lifetime in favor of proximity. The trade-off means that it is harder to find historical data. To quantify the effect, a simulation was run in which the simulation is divided into 5 epochs of 12 minutes each. Each query is generated with a time range limited to the first epoch. As the simulation progresses it becomes more and more difficult to find the original data. As shown in Figure 6 Locus's policies favor recent data over historical data, leading to lower query success rate for historic data.

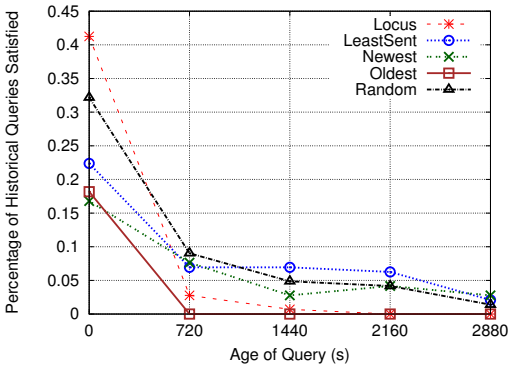


Figure 6: Success rates in satisfying historical queries.

6. CONCLUSION AND FUTURE WORK

The sensor and user-created data generated on end-user devices has the potential to enable a new class of live, location-based services for commuters, pedestrians and other mobile users. However, the large volumes of data these devices will generate cannot be supported by existing centralized approaches. Instead, we have designed a decentralized data overlay that runs on top of the mobile devices themselves. Because data producers and consumers are the same in this network, keeping the data near those devices brings greater utility to the network.

Our overlay, Locus, introduces the novel concept of "bubbles of knowledge" to keep data from a specific location near that location. By using location information, Locus can achieve more efficient data storage approaches and improve data look-up rates. As more users join the network, the benefits of Locus will be amplified due to the increasing density of the network, increasing both the performance and value of the overlay as the system grows.

One important factor in the performance of Locus is the size of the home area used to store data. In the current work a fixed size was used. An important avenue of our future work is to explore how to detect surrounding network conditions in order to dynamically adjust the size of data's home area.

In addition, the utility function used in the current implementation has a uniform shape for all data. However, the utility function could have very different shapes, depending on which kind of data they refer to and how far away from the home location the data is relevant. The utility functions can change dynamically in different environments and in response to changing application specifications and user interests. One application of dynamic utility functions is to enable caching of data at multiple locations in the network besides the home location. Utility functions can be defined that push data to different locations in the network, allowing the data to live nearer where queries are originating, reducing the forwarding needed to satisfy queries.

7. REFERENCES

[1] A. Balasubramanian, B. N. Levine, and A. Venkataramani. DTN routing as a resource allocation problem. In *Proc. ACM SIGCOMM*, 2007.

[2] M. D. Dikaiakos, S. Iqbal, T. Nadeem, and L. Iftode. VITP: an information transfer protocol for vehicular computing. In *International Conference on Mobile Computing and Networking*, 2005.

[3] J. Eriksson and S. Madden. Cabernet: vehicular content delivery using WiFi. In *Proc. ACM MobiCom*, 2008.

[4] V. Erramilli, M. Crovella, A. Chaintreau, and C. Diot. Delegation forwarding. In *Proc. ACM MobiHoc*, 2008.

[5] J. Harri, M. Fiore, F. Fethi, and C. Bonnet. Vanetmobisim: generating realistic mobility patterns for vanets. In *Proc. ACM VANET*, 2006.

[6] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: a distributed mobile sensor computing system. In *Proc. Conference On Embedded Networked Sensor Systems*, 2006.

[7] K. Ibrahim, M. C. Weigle, and M. Abuelela. p-IVG: Probabilistic Inter-Vehicle Geocast for Dense Vehicular Networks. In *Proc. IEEE VTC*, 2009.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. AMC CoNEXT*, 2009.

[9] S. Jain, K. Fall, and R. Patra. Routing in a delay tolerant network. In *Proceedings of ACM SIGCOMM*, 2004.

[10] B. Karp and H. T. Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proc. ACM MobiCom*, 2000.

[11] A. Keränen, J. Ott, and T. Kärkkäinen. The one simulator for dtn protocol evaluation. In *Simutools*, 2009.

[12] S. C. Nelson, M. Bakht, and R. Kravets. Encounter-based routing in dtns. In *Proc. IEEE INFOCOM*, 2009.

[13] M. Piórkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. On clustering phenomenon in mobile partitioned networks. In *Proc. ACM MobilityModels*, 2008.

[14] T. Spyropoulos, K. Psounis, and C. S. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. In *Proc. IEEE PERCOMW*, 2007.

[15] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11(1), 2003.

[16] B. Tang, H. Gupta, and S. R. Das. Benefit-based data caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 7(3), 008.

[17] N. Thompson, S. Nelson, M. Bakht, T. Abdelzaher, and R. Kravets. Retiring replicants: Congestion control for intermittently connected networks. In *Proc. IEEE INFOCOM*, 2010.

[18] A. Vahdat and D. Becker. Epidemic routing for partially-connected ad hoc networks. Technical Report CS-200006, Duke University, April 2000.

[19] H. Wu, R. Fujimoto, R. Guensler, and M. Hunter. MDDV: a mobility-centric data dissemination algorithm for vehicular networks. In *Proc. ACM VANET*, 2004.

[20] L. Yin and G. Cao. Supporting cooperative caching in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(1), 2006.

[21] X. Zhang, J. Kurose, B. N. Levine, D. Towsley, and H. Zhang. Study of a bus-based disruption-tolerant network: mobility modeling and impact on routing. In *Proc. ACM MobiCom*.

[22] J. Zhao and G. Cao. VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks. *IEEE Transactions on Vehicular Technology*, 57(3):1910–1922, 2008.