Evaluating (Geo) Content Sharing with the ONE Simulator

Michael S. Desta, Esa Hyytiä, Ari Keränen, Teemu Kärkkäinen, Jörg Ott
Department of Communications and Networking (Comnet), Aalto University
michael.desta@aalto.fi, {esa,akeranen,teemuk,jo}@comnet.tkk.fi

ABSTRACT

The Opportunistic Networking Environment (ONE) Simulator is an extensible tool for evaluating protocols and mobility models for delay-tolerant networking. ONE allows easily plugging in mobility models, contact traces, routing modules, applications, and report modules. In this paper, we describe how to instrument the ONE simulator for two content sharing applications: spreading content in waves across mostly stationary nodes and geo-based content sharing between mobile nodes using *Floating Content*. Both are included in the ONE simulator version 1.5.1.

Categories and Subject Descriptors

C.2.2 [**Computer Communication Networks**]: Network Protocols; I.6.7 [**Simulations and Modeling**]: Simulation Support Systems

General Terms

Design, Experimentation

Keywords

Delay-tolerant networking, simulation

1. INTRODUCTION

Mobile opportunistic networking may serve as important building block for exchanging information between users within the vicinity of each other. While such networks do not replace the Internet and mobile Internet access—which offer *global* communication and access to information and services irrespective of the locations of users and service providers—they are able to fill an important gap in enabling *local* communication. They come into play when using network infrastructure is not an option: because there is none, it is overloaded, too expensive, too restricted, or too slow.

Traditional Internet communication has focused on interactions between hosts. Quite a bit of the work on mobile opportunistic and delay-tolerant networking research has followed this idea, too, albeit opportunistic caching [12] and content replication [6] ideas appeared early, somewhat similar to the trends towards information-centric networking for infrastructure networks. However, delay-tolerant networking systems cannot compete with Internet-based services when user expectations demand instant responses, so that alternative ways for service interactions are required: as one example, DTN applications should operate in the background to collect information and execute services and present whatever "results" are available whenever user interacts with the application [8].

This approach works, for example, for content sharing applications such as *PodNet*, in which mobile nodes synchronize (parts of) their databases when in contact and the amount of content available to each user grows in the background. In other examples nodes "subscribe" to (web page) contents in the background (somewhat similar to RSS feeds) [1, 13, 14]. Yet another class of content sharing applications provide location-tagged content sharing as the server-less counterpart of digital graffiti [2]. Content items are associated with a location or an area for which they are valid and are replicated within this area whenever nodes meet, and deleted when nodes leave the validity area. Numerous systems were designed exhibiting different flavors of geo-content replication and access, including our work on *Floating Content* [9, 10], but also *Hovering Information* [16], Locus [15], and variants targeted at VANETs [5, 7]. All these sharing applications have in common that messages, i.e., content items, are not addressed to a particular node, but are "published" in the system together with descriptive metadata, such as a *channel* in PodNet or an *anchor zone* for Floating content.

This paper introduces the content sharing mechanisms in the 1.5.1 release of the ONE simulator supporting circulating content among fixed nodes (section 2) as well as geo-based content sharing among mobile nodes using *Floating Content* (section 3) along with a local open mobility model for local content sharing. We present the implementation and then discuss metrics for evaluating content sharing systems and visualizing content distribution using the ONE reporting functions in section 4, and conclude in section 5.

2. SHARING IN STATIC ENVIRONMENTS

Delay-tolerant content sharing may not just happen in sparse, dynamic environments but also in rather static setups such as in a stadium, in a park, or along a beach. Such environments are characterized by a) fairly static nodes that may come and go over time, b) potential disruptions due to gaps that nodes leaving or sleeping create, and c) large dimensions that prevents direct communication between all nodes. Assuming that more content exists than a single node would store, one option is to make the content circulate periodically in waves (like *la ola* in a stadium) through all the nodes. Any newcomer would then get access to all contents over time.

The metrics of interest for this system include the basic availability of a piece of content (#copies), which also indicates resource

MobiWac'13, November 3–8, 2013, Barcelona, Spain.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2355-0/13/11 ...$15.00
http://dx.doi.org/10.1145/2508222.2512839.

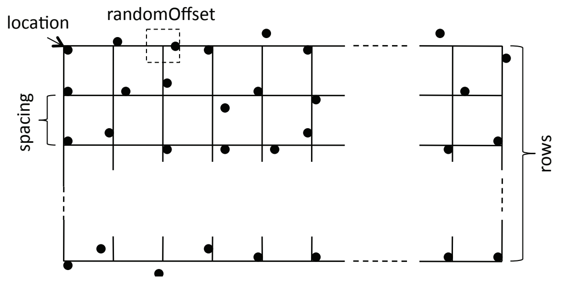

Figure 1: GridLocation "Mobility" Model

consumption; the time to obtain a copy of the content for a node; and the distribution of the messages. We report on the metrics further in section 4. We evaluated such content sharing models in [11], for which we designed a static grid mobility model and two routing algorithms that are now included in the ONE simulator.

2.1 Mobility

The GridLocation class offers a simple way of specifying nodes on a grid. As shown in figure 1, the grid is defined by the (location) of the upper left node in the grid so that it can be placed anywhere in the ONE world, the number of rows, and the (mean) spacing between nodes. The number of columns is calculated from the node count. A randomization factor (randomOffset) may be specified for the node positions to make the grid less regular: node locations are then spread around the grid coordinates uniformly within $\pm$randomOffset.

2.2 Routing

Two routing modules make content circulate across the nodes:

2.2.1 *Game of Life Router*

Since the nodes in our setup are mostly static, we view them as cells the Game of Life automaton [4]. Game of Life simulates cells by a set of simple rules that infer the fate of a cell in a given position X from the number of live adjacent cells: if there are too few (m) or too many neighbors (n), the cell dies; for some range of neighbors a new cell is born; and for another range, a cell survives.

The LifeRouter implements a simplified model of the custom Game of Life simulator used for part of the evaluation in [11]. The LifeRouter determines neighboring cells via the radio range, to which it is thus connected. The router has one parameter that governs its behavior, nmcount, that defines the pair n, m. During the operation, if a node does not have a copy of the message and it is connected to k nodes, $m \leq k \leq n$, the node will obtain a copy of the message; if the node already has a copy of the message, it will keep it. If $k > n \vee k < m$, the message will not be replicated to a new node. Instead, it will be deleted on a node that has a copy.

With the right parameter choices, messages are spread in cycles in parts or throughout all of the network.

2.2.2 *Wave Router*

The WaveRouter is a routing scheme to make messages circulate. To achieve this, each node receiving a message, forward and then stores it for a while before deleting it. After the deletion, the node does not accept the same message for a while. When all nodes act this way, the messages move through the grid in waves.

The operation is controlled by one internal data structure and two parameters. Each node keeps track of the messages it has recently received in a *tracking list*. The messages are kept in the list for a certain fixed time period—the immunity time (immunityTime)—during which the node is "immune" to them and will not accept those messages again. This basically ensures that messages do not bounce back to the same direction they came from so that waves can form. A second parameter (custodyFraction), which is measured relative to the immunity time, defines how long a node keeps a copy of the message after receiving it. During this time, the content is available for local application and for replication to newcomers as well as neighbors losing their immunity.

The message deletion is executed independently of the tracking list, i.e., the node may be immune to a message even though the message is no longer in its possession. Moreover, upon accepting a message, the node agrees to take over the custody of the message, which means that it will not delete the message until it has been passed further (along with the custody) at least once or the message's time-to-live expires.

3. FLOATING CONTENT

Floating Content is a content sharing scheme for mobile users. It uses a simple system model for its operation, assuming that the mobile nodes are aware of their locations [9]: Node S creates a data item I at a given location $L = (x, y)$. I is associated with two radii, r and a, that define the distances from L within which I will be replicated to other nodes encountered (r), and within which I remains valid (a), i.e., a node will keep its copy; and with a lifetime T after which I will be deleted. Items will also be deleted outside this *anchor zone* delineated by a.

The system offers one communication primitive: posting content into an area from which it may be picked up by other nodes. The operation is inherently best-effort and asynchronous: if all nodes carrying a copy of an item leave the anchor zone, the data item disappears; and there is no upper bound (besides T) within which a data item might reach other nodes. Content is picked up in the background and inserted into a local database, from where it can be retrieved via application queries [10]. Content cannot be deleted (except by expiration).

The *Floating Content* implementation in the ONE simulator comprises three modules: the routing module, the application module, and the reporting module for floating content. They can be used with arbitrary mobility models, but we developed a specific open mobility model for urban squares, which we describe first before turning our attention to the floating content implementation.

3.1 Square Mobility Model

Areas where geo-based opportunistic content sharing can be envisioned, such as city squares, show high dynamic properties especially in terms of node density. This property is mainly due to the "openness" of the area, which allows nodes to freely enter and depart the system rather than having a fixed set of nodes confined in a closed system. A mobility model that captures this property of openness, the *Square Mobility Model*, is introduced in [3].

The Square Mobility Model is defined as follows:

- Nodes move in $a \times b$ rectangle area $\mathcal{A}$.
- New nodes enter the area through one of the sources located at each corner of the rectangle and move freely in $\mathcal{A}$.
- Upon reaching a waypoint in $\mathcal{A}$:
  - With probability of $p_i^{(\text{exit})} = P^{(\text{exit})}/4$, $i = 1, \ldots, 4$, the node moves next to a exit point (sink) i, where it then departs the system (sinks and sources are co-located).
  - Otherwise, it chooses a new waypoint uniformly in random from $\mathcal{A}$ and starts moving directly towards it.
- Velocity v for each transition is chosen independently from an arbitrary velocity distribution with $\mathrm{E}[1/v] < \infty$.

In short, nodes enter the square through one of the four corners of the square, then move according to RWP for a random number of transitions, and then depart the system.

Parameter $P^{(\text{exit})} \in [0, 1]$ controls the sojourn behavior of the nodes. Areas where people tend to spend more time moving around, e.g. flee markets, can be characterized with a lower exit probability, $P^{(\text{exit})}$, whereas busy intersections in cities have a much higher exit probability. Setting $P^{(\text{exit})}{=}0$ gives us the RWP movement.

We have implemented the *Square Mobility* model by adding a new movement class, the SquareMobility. Since one of the main properties of open city squares is that nodes can enter an area of interest according to some arrival rate, e.g. Poisson, move freely inside the area, and depart when they choose so. The SquareMobility module introduces a feature to add nodes to and remove them dynamically from the simulation. Dynamic addition and removal of nodes can be configured by setting the static variable dynamic from the node creation event of a host group as true, whereas the default value or the setting HostGroup_NS.dynamic=false disables this feature. A user can also configure the inter-arrival time distribution of nodes by setting up the static variable distribution for the event NodeCreationEventGenerator to the desired value, where poisson and uniform are currently available. Hence, for dynamic host groups, the user needs to configure the arrival rate of nodes, e.g. the intensity for Poisson arrival rate using another static variable lambda with a desired integer value.

3.2 Floating Content Routing

The FloatingContentRouter implements the replication and deletion operation as described in [9]. Besides the buffer size (inherited like other parameters from the ActiveRouter class, four parameters govern its operation: the 1) replicationAlgorithm and 2) deletionAlgorithm govern the behavior when nodes encounter each other outside the replication area but within the availability area of a content item. Simulations suggest making this part of the anchor passive, i.e., neither replicating nor deleting content [9], which is achieved by setting both to none.

3) The replicationPolicy determines the order in which floating messages are replicated when two nodes come into contact. While FIFO and random ordering policies could be used, we found STF2 (see [9]) to be very useful, as it prioritizes messages inversely proportional to their resource consumption, defined by the product of anchor zone area, message size, and lifetime. Of course, there is room for inventing and experimenting with more subtle policies.

4) The deletionPolicy defines when a message is purged from a nodes outside the anchor zone. We find that rather than continuously checking the location against stored messages to enable immediate deletion, waiting until the encounter with another node is more practical, has no practical drawbacks, and is closer to real implementations. Note that the above policies are not applied when choosing which messages to purge to make room for new incoming messages. Instead, we rely on the functions of the ActiveRouter.

All parameters specific to floating content are attached to the ONE messages as *properties*, the name space shared between the router, the applications, and the reporting modules.

3.3 Applications and Reporting

The FloatingApplication defines when and how floating content items are generated. At present, the application for evaluating Floating Content itself is included in the distribution; the generic application framework and API [10] is still under development to make it more generic.

The implementation uses the application framework of the simulator: nodes are associated with the application as potentially one out of several, and different (groups of) nodes can have different application instances assigned. The parameters are then defined per application instance.

Four parameters determine message generation: startTime indicates when message generation starts, interval specifies the minimum time between two consecutive messages generated per node, and ttl and messageSize define the lifetime and the size of the messages. For both, the values are passed as triples: min, max, gran, which define the minimum and maximum value and the granularity in which values are chosen uniformly in random. For example, setting $1800, 7200, 900$ for ttl would choose random message TTLs between 30min and 2h in 15min steps. For historic reasons, the anchor zone size parameters r and a are specified separately.

By default, the entire simulation world would be chosen as anchor points for floating content items. A rectangular area can be specified via minimum, maximum, and step size as in above via the parameters anchor, anchorMax, and anchorGran. Each takes a quadruple as input, defining first the X- and Y-coordinates, and then r and a. Indicating an anchor granularity greater than one for the X- and Y-coordinates creates a grid, the points of which become legitimate anchor points. Nodes will only generate floating content item when with the specified coordinates, the messages may "snap to the grid point", and will choose anchor zone sizes uniformly distributed between the respective minimum and maximum values for r and a at the given step size.

The FloatingApplication also supports a mode of operation with a single anchor zone, specified in the configuration file using the above parameters and setting mode=fixed (the default is variable). In this case, mobile nodes entering the anchor zone and getting sufficiently close to the anchor point (inAnchorZoneFactor) may post messages with the indicated anchor point. This mode can be used to systematically assess the floating probability at individual locations on a map [9]. For such a single anchor zone, also the entering and leaving nodes can be counted by setting the parameter flux=true and using the FloatingAppReporter for reporting.

To evaluate a specific application, a custom application class implementing the respective functionality (such as the auction in [10]) needs to be developed.

Besides the above FloatingAppReporter, which is used only for a very specific purpose, the FloatingMessageReport provides specific details on floating content items: it reports their *create* and *delete* events, whenever a replication from one node to another is *start*ed and then either *abort*ed or the message is successfully *replicate*d. Along with each event, time stamp, node, message id, position, and all floating and message parameters are logged.

In conjunction with the *Square Mobility* model, we have added two more reporting classes; the FloatingDurationReport class and the NodeSojournReport class, which are used to collect data about content floating duration and the mean sojourn time of a node, respectively. The FloatingDurationReport reporting class works by keeping track of the latest time a given piece of content was relayed before it "sinks" or the simulation ends and report back the value. In the NodeSojournReport class, the specific time a node is removed from the simulation is recorded by the nodeEnd() method which is called when a node destruction event is executed. This time is then used by another method, getNodeLifeTime(), to calculate the node's sojourn time. In situations where the nodeEnd() method is not called, e.g. when the simulation time ends, then the getNodeLifeTime() returns the difference in the simulated seconds elapsed from the creation of a node until the end of the simulation. In other cases, e.g., when the simulation is interrupted by a user, the current simulation time will be used.

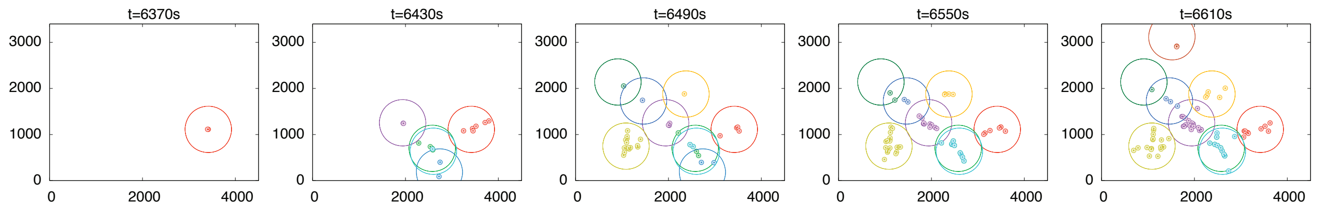

Figure 2: Floating Content visualization in 60s intervals with different colors for the content items: the large circles show the anchor zones, the nodes and their radio ranges are shown as dots with small circle. The X and Y axes show location offsets in meters.

4. METRICS AND VISUALIZATION

As mentioned above, important metrics for content sharing include 1) if and how long a content item remains available; 2) how the content is (geographically) spread; 3) how effectively the content is distributed to the (interested) nodes, i.e., how many nodes that should obtain the content actually receive it; 4) how long it takes for an eligible node to receive a copy; and 5) how many copies of a content exist in the system.

The ONE simulator offers a number of generic and custom report modules whose output either answers the above questions or can be combined by scripting to obtain the desired results. There is, however, one caveat for content sharing. Messages in ONE required a destination host they are addressed to. But the content sharing systems presented in this paper don't have explicit receivers of the message. To avoid that messages stop circulating when they reach the "destination", we choose a node that is inactive and hence unreachable for as the target. Replication and delivery to the local application is dealt with by the respective routing module using other parameters (e.g., by matching subscriptions against message metadata) than the destination address. This implies that ONE reports on message delivery cannot be used and separate reporting mechanisms are necessary.

The MessageCopyCountReport allows tracking the total number of copies of each message within the system and thus can answer (1) and (5), where the MessageLocationReport records the locations of all those copies and thus answers (2). Both report in regular intervals. In contrast, the FloatingMessageReport provides event-driven tracking of all generated messages and yields precise time and location information and allows inferring how long messages stayed available and when they were replicated to which entity. With appropriate scripting, this allows answering (1) and (5) at a finer granularity. For Floating Content, using the FloatingAppReporter allows determining the time it takes a mobile node to obtain a copy of a message since entering the anchor zone (and which of those nodes do not get a copy at all). This provides answers to (3) and (4).

Combining the new FloatingMessageReport class together with the MessageLocationReport class aids visualizing the spread (and demise) of pieces of content. An example is shown in figure 2 (see http://floating-content.net/anim.html for an animated version). The message creation history was obtained from the former report and the locations were recorded every ten seconds using the latter. The reports were filtered for a subset of messages, in this case the generated messages #10–#19. Then the messages and their anchor zones were plotted using *gnuplot* in different colors, message copy shown with its radio range. The figure shows a time series with five snapshots. We can see how content items are created. Some of them are replicated frequently across a number of nodes and will stay available, while others disappear again quickly (as the one around coordinate $(0, 2800)$ intersecting with the Y axis in $t \in \{6430, 6490\}$s and likely then one at $(3000, 1800)$ at $t = 6610$s.) Using GIF as an output format allows creating animations from the individual messages with tools such as *gifsicle* and *ImageMagick*.

5. CONCLUSION

In this paper, we have described support for simulating and evaluating different content distribution schemes in the ONE simulator. The new release including these (and other new) features, ONE v1.5.1, will be available for download at the ONE simulator homepage: http://www.netlab.tkk.fi/tutkimus/dtn/theone/

Acknowledgments

This work was supported by the Academy of Finland in the PDP project (grant no. 260014), and by the European Commission within EIT ICT Labs.

6. REFERENCES

[1] A. Balasubramanian, Y. Zhou, W. B. Croft, B. N. Levine, and A. Venkataramani. Web search from a bus. In *CHANTS '07: Proceedings of the second workshop on Challenged networks CHANTS*, pages 59–66, 2007.

[2] S. Carter, E. Churchill, L. Denoue, and J. Helfman. Digital graffiti: public annotation of multimedia content. In *Conference on Human Factors in Computing Systems*, pages 1207–1210, 2004.

[3] M. S. Desta, E. Hyytiä, J. Ott, and J. Kangasharju. Characterizing content sharing properties for mobile users in open city squares. In *10th Annual IEEE/IFIP Conference on Wireless On-Demand Network Systems and Services (WONS)*, Mar. 2013.

[4] M. Gardner. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, pages 120–123, 10 1970.

[5] E. Koukoumidis, L.-S. Peh, and M. Martonosi. RegReS: Adaptively Maintaining a Target Density of Regional Services in Opportunistic Vehicular Networks. In *Proc. IEEE PerCom*, 2011.

[6] V. Lenders, M. May, G. Karlsson, and C. Wacha. Wireless ad hoc podcasting. *ACM/SIGMOBILE Mobile Comp. and Comm. Rev.*, 12(1), 2008.

[7] B. Liu, B. Khorashadi, D. Ghosal, C.-N. Chuah, and M. Zhang. Assessing the VANET's local information storage capability under different traffic mobility. In *Proc. IEEE Infocom*, 2010.

[8] J. Ott. 404 Not Found? – The Quest for DTN Applications. In *Keynote abstract. Proc. of MobiOpp 2012*, March 2012.

[9] J. Ott, E. Hyytiä, P. Lassila, T. Vaegs, and J. Kangasharju. Floating Content: Information Sharing in Urban Areas. *Elsevier Personal Wireless Communications (PMC)*, 7(6):671–689, 12 2011.

[10] J. Ott and J. Kangasharju. Opportunistic Content Sharing Applications. In *Proc. ACM MobiHoc NOM workshop*, June 2012.

[11] J. Ott, A. Keränen, and E. Hyytiä. BeachNet: Propagation-based Information Sharing in Mostly Static Networks. In *Proc. of ExtremeCom*, 2011.

[12] J. Ott and M. Pitkänen. DTN-based Content Storage and Retrieval. In *The First IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC)*, June 2007.

[13] M. Pitkänen, T. Kärkkäinen, and J. Ott. Opportunistic Web Access via WLAN Hotspots. In *Proceedings of the 8th Annual IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 3 2010.

[14] G. Sollazzo, M. Musolesi, and C. Mascolo. TACO-DTN: a time-aware content-based dissemination system for delay tolerant networks. In *Proc. MobiOpp '07*, pages 83–90, 2007.

[15] N. Thompson, R. Crepaldi, and R. Kravets. Locus: A location-based data overlay for disruption- tolerant networks. In *Proc. ACM CHANTS*, 2010.

[16] A. Villalba Castro, G. Di Marzo Serugendo, and D. Konstantas. Hovering information: Self- organizing information that finds its own storage. Tech. Rep. BBKCS707, Birkbeck College, 2007.