Shared Content Editing in Opportunistic Networks

Teemu Kärkkäinen
Aalto University, Comnet
Espoo, Finland
teemu.karkkainen@aalto.fi

Jörg Ott
Aalto University, Comnet
Espoo, Finland
jorg.ott@aalto.fi

ABSTRACT

This paper examines shared content editing in opportunistic networks. Instead of immutable messages, such as photos or music files that are often assumed in opportunistic network applications, we focus on mutable content, such as wiki-pages, that can be edited by anyone carrying a copy. We show through simulations that mutable content can be handled by using revision control mechanisms (merging), or by simply adopting/discarding versions wholesale.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Store and forward networks*

Keywords

Mutable content; opportunistic networking; version control

1. INTRODUCTION

Opportunistic networking has enabled content sharing and distribution in challenging environments. Whether this content is a page from a web server [6], picture from a camera or a Twitter message by the user [2], the content itself is immutable and explicitly owned by its creator. Other nodes may consume or forward the content, but not modify it. However, many popular distributed applications – such as message boards, discussion forums and wiki pages – deal with mutable content that many users may edit simultaneously. This brings up the problem of maintaining a consistent content state in the face of possibly conflicting modifications.

In classical distributed applications with mutable content this problem is typically solved by having a dedicated server act as a gate keeper to the data, enforcing rules that guarantee consistency (e.g., a wiki server making sure edits are applied sequentially without conflicts). In opportunistic networks without guaranteed connectivity to infrastructure, disconnected nodes spread around the environment carry their own copies of the content and may modify those copies at any time. Since every local modification creates a new version of the content item (and a new message in the network to carry it) the network is quickly congested with messages carrying slightly different versions of the same logical content. This clearly calls for mechanisms to stop the version explosion by ensuring that every node carries only one copy of each content item, while still being able to receive modifications made by other nodes.

In this paper we present two main approaches to handling mutable content: *adoption*, and *merging*. The first is the simpler model; when two nodes with different versions of the same content item meet, they can choose to adopt one of the versions while discarding the other. The second is more complex; the nodes attempt to generate a new version that includes changes from both the versions by merging the content items. These mechanisms can be fully automatic, or include user interaction to help in the process. We discuss each of these in more detail in Section 2 and present analysis of their performance and behaviour based on simulations in Section 4.

There is a wealth of prior work on opportunistic content dissemination and applications [4]. Recently content or information centric approach has been applied to opportunistic networks for content dissemination [1]. However, these approaches build heavily on the assumption of immutable content, e.g., for security and caching mechanisms, and are at odds with collaboratively editable content.

There is also a long history of research into collaborative content editing mechanisms in fixed networks [11]. More recently Commutative Replicated Data Types have been used as a basis for collaborative editing systems, including peer-to-peer systems, in mostly theoretical work [7, 12]. A practical way of enabling collaborative document editing in classical networking scenarios, including mobile nodes, based on a weakly consistent replication platform is presented in [8]. While this approach includes direct peer-to-peer synchronization, it does not assume an opportunistic network and is rather based on traditional networking scenarios. In contrast, our work assumes a fully opportunistic and distributed network with no central repositories and each node having its own authoritative version of the content.

2. SHARED CONTENT EDITING

Our system assumes independent mobile nodes that form opportunistic contacts between each other when within short range communications (e.g., Wi-Fi). Each node carries copies of content items, one copy of each item, which it may modify at will by adding or deleting parts. The copies contain the current content as well as a full history of previous versions (similarly to software version control systems). Upon encountering each other, nodes will attempt to form a consistent view of their two content versions. This corresponds to, e.g., mobile nodes running some opportunistic middleware or Bundle Protocol [10] routers with a shared content editing.

There are two main approaches that can be taken when encountering a modified version of the content item: 1) adopt (or discard)

CHANTS'14, September 7, 2014, Maui, Hawaii, USA.

ACM 978-1-4503-3071-8/14/09 ...$15.00.
http://dx.doi.org/10.1145/2645672.2645685.

the modified version, or 2) attempt to merge the two versions into a new version that includes the changes contained in both. The first approach, adoption, will discard messages and therefore the changes to content contained in them. The second approach, merging, will retain all the changes, but may result in merge conflicts where two versions cannot be combined.

We model the modifications to a content item that is represented by an ordered list of elements. The content can be modified by removing an element from a given position in the list, or adding an element to some position. The only requirement for the elements is to be comparable for equality with other elements. The elements map into different semantic units in different real applications. In a wiki, for example, the elements would map to words of text. Other types of structures for the content items, for example trees, are possible and require different merge semantics, but they are out of scope for this work.

2.1 Merging

The versioning model is distributed, with each node holding its own authoritative copy of the content item and its full version history. When meeting, nodes attempt to synchronize their copies as follows: 1) If the peer's version is a direct ancestor of the local version, it is ignored. 2) If the peer's version is a direct descendant of the local version, the node will adopt the peer's version. 3) Otherwise a merge is attempted, which can result either in a conflict or in a new version that combines the local and peer's versions.

The merging of the content could be done manually by the user, but can in many cases be also done algorithmically. There are multiple approaches to automatic merging, including *three-way merge*, *weave merge*, and *patch commutation*. Different algorithms have differences in operation and required input. We focus on the three-way merge, which is widely used in version control systems.

Three-way merge is calculated using the full version history of the content being merged. The history can be represented as a directed acyclic graph (DAG), where each vertex corresponds to a unique version of the content, and edges correspond to changes in the content. In the case of a modification of the content, there will be one edge from the old version to the new one. In the case of a merge, there will be edges from the two parent versions to the merged version. The DAG is constructed by combining the local version history with the peer's version history.

The merge is done by first finding the lowest common ancestor (LCA) for the two versions in the DAG. The LCA corresponds to the content version from which the two merged versions branched off from. The difference between the local version and the LCA, and the peer's version and the LCA is calculated, resulting in two sets of changes (represented as removals and additions to the LCA content). These two sets of changes are then combined and applied to the LCA in order to produce the new merged version.

The merge will succeed if all the changes in the two change sets either apply to different positions in the LCA, or both sets contain the same change to the same location. A merge conflict occurs when both change sets contain an addition of a different element to the same location. Resolving conflicts requires understanding of the content semantics, and therefore cannot be safely automated.

3. SIMULATIONS

We evaluate the behaviour and performance of the shared content editing mechanisms through simulations using The ONE simulator [3] in a range of scenarios. The simulation scenarios and the content editing model is described in this section, while the results are presented and analysed in the next section.

3.1 Simulation Scenarios

We consider three scenarios: 1) static and well connected, 2) random movement, and 3) realistic trace-based movement. The static and random movement scenarios are simple and cover the extremes of mobility found in opportunistic networks. However, due to their simplicity these scenarios do not represent real-world opportunistic networks well. We therefore include a trace-based movement model to evaluate the behaviour in a realistic scenario. For the static scenario we construct a hexagonal grid topology out of 61 nodes so that each node has six neighbors (except along the edges). We use Modified Random Direction [9] movement model for random movement with 60 nodes, $1ms^{-1}$ speed, $(0, 120)sec$ pause time, $1 \times 1km$ area, with $30m$ radio range over 20000 simulated seconds. We avoid the widely used Random Waypoint Model due to its known poor characteristics such as uneven density and failure to reach a steady state [9, 13]. For the realistic scenario we use the SMOOTH mobility model for 7 days with parameters derived from a real trace taken from the KAIST campus [5][1]. In all cases we assume that communication delays, link capacities, and node buffers are insignificant and will not be a bottleneck.

3.2 Content Editing Model

We simulate a pure *merging* model, and pure *adoption* model. We analyse the growth of a content item initially composed of 100 "lines". Every node starts with a copy of the content item and modifies it periodically by inserting 5 random lines in random positions. In the first two cases the nodes modify the content on average every 1000 seconds, while in the third, realistic scenario the nodes modify the content once per 30240 seconds. For the first two scenarios these values are chosen to clearly expose the fundamental characteristics of the content modification process, while in the last one they are chosen to reflect a realistic case of a wiki-like page that is infrequently added to by the users. Both cases result in each node modifying their local content 20 time over the simulation period. For simplicity, we do not consider deletions, noting that a deletion cannot directly conflict (although they can solve conflicts by removing conflicting lines).

We model *destructive* and *non-destructive* conflict resolution for merging. In the *destructive* case the node selects the full set of changes for each conflict from one of the versions and discards the changes from the other. We further assume that the users always make consistent choices, i.e., given the same conflict all users always solve it the same way. We do not consider the case where users cannot choose consistently, either by having different users choosing differently (e.g., "edit wars") or by users changing their choice over time. In particular, we assume that the conflict is solved by taking the larger of the conflicting change-sets, or in the case of same sized sets picking arbitrarily but consistently. In the *non-destructive* case the node retains all the changes from both versions by choosing to apply the full change-set from the first version followed by the second version. The order is again consistent, with all nodes applying the changes in the same order.

For the adoption model we consider two decision making criteria when choosing which one of two versions to take and which to discard: 1) choosing the version with *latest* changes, or 2) the *longest* content item.

4. RESULTS AND ANALYSIS

This section presents the results of the simulations and their analysis. First the characteristics of a pure merge strategy are studied

[1]All parameters are taken from the cited paper.

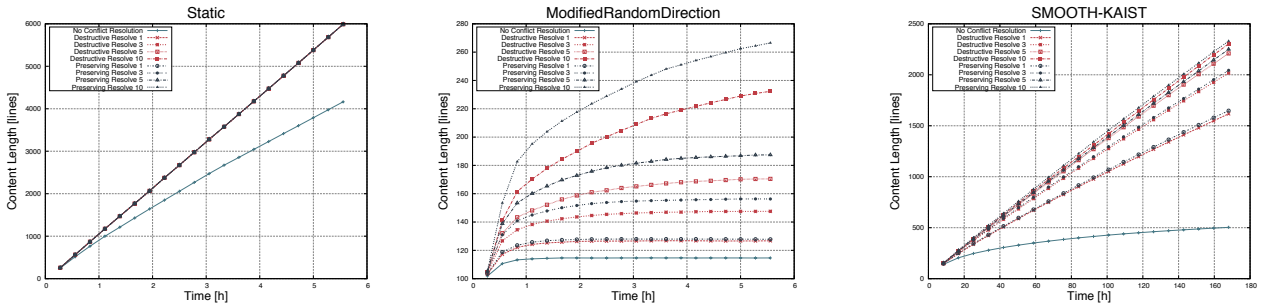

Figure 1: Length of content over time for merging strategies.

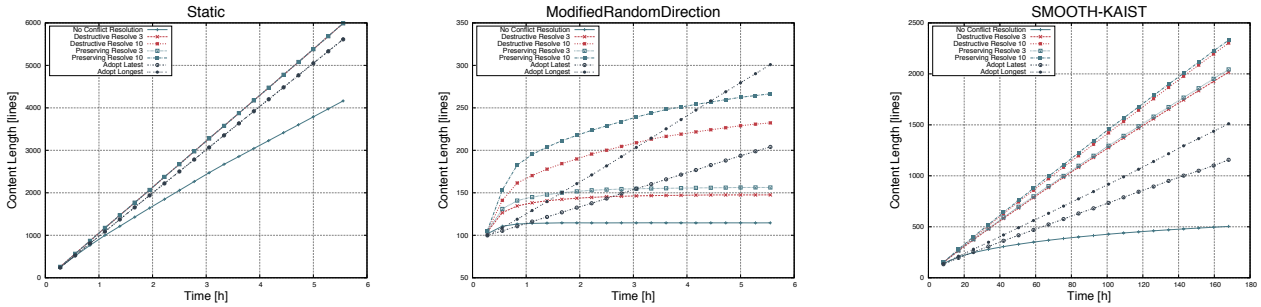

Figure 2: Length of content over time for adoption strategies (merge strategies as reference cases).

in the various scenarios. Then the performance of the pure merge strategy is compared with the simple adoption strategies.

4.1 Merging Strategy

We first examine the growth of a content item when applying a pure merging strategy. Figure 1 shows the contribution of the merging mechanism to the mean length of the content over time in the three different scenarios. The impact of the nodes modifying their local content items has been subtracted out of the average length leaving only the impact of changes merged from other nodes. The results are shown for both *destructive* and *preserving* merge strategies with the ability to resolve conflicts of various sizes, along with the case where no conflict resolution is done.

In the *static*, well connected scenario on the left of the figure, it can be seen that all the strategies with any conflict resolution perform equally well; they all achieve practically optimal performance where every node sees every change. This is expected since the changes propagate quickly between the nodes and the likelihood of a conflicting changes is small since the nodes have an up to date view of the global changes. However, the propagation of the changes still takes some time and it is still possible that conflicts are introduced. This can be seen from the behaviour of the case where no conflict resolution is possible, where the performance is worse compared to the other cases. As can be seen from the difference in the graphs, the impact of even a small number of unsolvable conflict cases is significant. This is due to the speed at which the conflicting versions propagate in a well connected scenario. In theory a single conflict could split the node population into two halves carrying conflicting versions and thus being unable to merge changes from each other, effectively splitting the merging rate in half. In practice the impact is not quite that drastic since the population is unlikely to be split in two equal halves (one of the conflicting changes will be made slightly earlier and thus propagate more widely). Fortunately this effect can be countered with the ability to solve even a single conflict at a time, resulting in near optimal performance. Whether the conflict resolution preserves or discards changes is insignificant, since the number and size of conflicts is so small that it has trivial impact on the overall performance.

Middle of Figure 1 shows the *Modified Random Movement* scenario where nodes are not well connected, but instead move randomly and attempt to merge their content versions when meeting. In this scenario the changes made by nodes must propagate to other nodes through a series of contacts and merges. Further, the changes must propagate along a chain of nodes not carrying versions with unsolvable conflicts. Essentially the reach of a change is limited to the set of nodes with non-conflicting versions, and by the epidemic propagation within that set. Due to the simulation setup, once an unsolvable conflict is introduced, it will remain forever and therefore the sets of nodes that can potentially merge a given change decreases over time until reaching zero once all the nodes are carrying conflicting versions. In practice such conflicts could disappear through subsequent modifications where one or both of the conflicting changes are deleted from later versions, but that effect is not modelled in this work.

The resulting behaviour in the content length increase in the MRD case differs significantly from the well connected case. In all cases the rate of content increase goes towards zero, meaning that eventually no changes from peers are merged into the content Most of the length growth happens early, after which the growth rapidly ends. The limit is dependent on the number of conflicts that can be solved. When no conflicts can be solved, the limit is only about 15% above the original length of the content. If the nodes have the ability to solve conflicts, the limit increases up to 160% of the original content length within the simulation period. This is natural since the limit is reached when all live version of the content are conflicting with each other. Conflict solving slows down the rate at which the versions become conflicting, therefore extending the time during which merging can occur and thus increasing the limit that the content length can reach.

The MRD scenario also reveals differences in the destructive and preserving merge strategies. As expected, the change preserving merges lead to longer content lengths than the destructive merges that discard changes. The content preserving merge performs com-

paratively better when larger conflicts can be resolved. This is natural since the larger the solvable conflict, larger the number of solved conflicts and more changes are preserved by the preserving resolution mechanism and discarded by the destructive resolution.

The more realistic *SMOOTH-KAIST* scenario is presented in the right of Figure 1. The movement model results in clusters of high connectivity with some flights between the clusters. This corresponds to a hybrid between a well connected case (approximated in this work with the static scenario) and the fully opportunistic case (approximated by MRD). This is also reflected in the results; there is a significant difference in performance between the case with no conflict resolution and the case where a single conflict merges can be solved. This can be explained with the reasoning presented in the case of the static scenario, where inside the well connected clusters the introduction of unsolvable merge conflicts has a large impact and being able to solve even a small number of conflicts has disproportionately large positive impact on the performance. However, like the fully random scenario being able to solve larger conflicts improves the performance of the system. As the size of the solvable conflict increases, performance of the system approaches a steady state; in this case about 14 additional lines per hour, while the optimal rate is about 35 lines per hour. This corresponds to about 40% of the optimum performance where every node would receive every change. This behaviour is due to the clusters generated by the scenario being largely isolated, causing versions inside the clusters to remain non-conflicting, but versions in different clusters diverging and having little cross pollination of changes between them.

4.2 Adoption Strategy

Figure 2 shows the performance of adoption strategies as compared to the previously presented merge strategies. Two decision making approaches are shown; preferring the version with larger content and the preferring the version with more recent changes.

In the *static*, well connected scenario both adoption strategies perform identically; better than merging with no conflict resolution, but failing to reach the optimal performance of merge strategies with conflict resolution. Adoption strategies do not reach optimum performance since they discard changes in cases where a merging strategy is able to combine them. In particular, any change made to an old version will lose out to changes made to newer versions. As can be seen from the graph, the impact of this effect is not large in the well connected case where the new versions propagate quickly to the whole network.

In the *MRD* scenario, middle of Figure 2, the difference in the fundamental behaviour of the merging and adoption strategies can be seen clearly. While the merging strategies lead to aggressive growth early on, the growth quickly stops and the content length reaches a limit as unsolvable conflicts accumulate in the network. Adoption strategies on the other hand cannot lead to conflicts and therefore perform at a steady rate over time. It can be seen that early on the merging strategies are clearly superior to adoption strategies in distributing changes to the content, but eventually the adoption mechanisms overtake the stalling merging mechanisms. This indicates that in disconnected scenarios with random movement, simple version adoption strategies are superior to merging with or without conflict resolution.

While adoption based strategies eventually outperform merging strategies in random scenarios, in the more realistic *SMOOTH-KAIST* the situation is again reversed. While adoption strategies significantly outperform merging with no conflict resolution mechanisms, when conflict resolution capability is added merging consistently and significantly outperforms adoption. However, since merging with conflict resolution requires user interaction while adoption strategies do not, and since adoption strategies still perform relatively well, they are also viable in realistic scenarios.

5. CONCLUSIONS

In this paper we presented the problem of shared content editing in opportunistic networks and presented approaches for enabling it. These approaches include both merging of the content items and wholesale adoption of newer versions of the content. We evaluated these approaches in a range of scenarios from static to fully random movement, including a realistic trace-based model.

The results show that in a realistic scenario simple adoption based strategies perform surprisingly well, but that merging strategies are still superior if user intervention to solve small merge conflicts is possible. However, implementing a real shared content editing system and applications for opportunistic networks presents many further challenges, especially in the area of user interaction with the merging process.

Many aspects of the problem can be further studied, including the impact of delayed or aggregated merge conflict solving tasks, strategies that combine both merging and adoption, and many optimizations are possible including heuristics for making better choices on which versions to adopt and automatic merge conflict resolution when the content semantics are known (e.g., merging message boards based on timestamps). These are left for further study.

Acknowledgements

This project has received funding from the European Community's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 611366 (PRECIOUS).

6. REFERENCES

[1] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (netinf) âĂŞ an information-centric networking architecture. *Computer Communications*, 36(7):721 – 735, 2013.

[2] T. Hossmann, P. Carta, D. Schatzmann, F. Legendre, P. Gunningberg, and C. Rohner. Twitter in disaster mode: security architecture. In *Proceedings of the Special Workshop on Internet and Disasters*, SWID '11, pages 7:1–7:8, New York, NY, USA, 2011. ACM.

[3] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE Simulator for DTN Protocol Evaluation. In *SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, New York, NY, USA, 2009.

[4] J. Leguay, A. Lindgren, J. Scott, T. Friedman, and J. Crowcroft. Opportunistic content distribution in an urban setting. In *ACM SIGCOMM workshop on Challenged networks (CHANTS)*, pages 205–212, New York, NY, USA, 2006. ACM Press.

[5] A. Munjal, T. Camp, and W. C. Navidi. Smooth: A simple way to model human mobility. In *ACM MSWiM*, 2011.

[6] M. Pitkanen, T. Karkkainen, and J. Ott. Opportunistic web access via wlan hotspots. In *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pages 20 –30, April 2010.

[7] N. Preguica, J. M. Marques, M. Shapiro, and M. Letia. A commutative replicated data type for cooperative editing. In *IEEE ICDCS'09*, 2009.

[8] K. P. Puttaswamy, C. C. Marshall, V. Ramasubramanian, P. Stuedi, D. B. Terry, and T. Wobber. Docx2go: Collaborative editing of fidelity reduced documents on mobile devices. In *ACM MobiSys*, 2010.

[9] E. Royer, P. Melliar-Smith, and L. Moser. An analysis of the optimum node density for ad hoc mobile networks. In *IEEE ICC*, 2001.

[10] K. Scott and S. Burleigh. Bundle Protocol Specification. RFC 5050, November 2007.

[11] C. Sun and C. Ellis. Operational transformation in real-time group editors: Issues, algorithms, and achievements. In *CSCW '98*, 1998.

[12] S. Weiss, P. Urso, and P. Molli. Logoot: a scalable optimistic replication algorithm for collaborative editing on p2p networks. In *IEEE ICDCS'09*, 2009.

[13] J. Yoon, M. Liu, and B. Noble. Random Waypoint Considered Harmful. In *Proceedings of IEEE Infocom*, April 2003.