Exploiting Synergies Between Coexisting Overlays

Shen Lin, François Taïani, and Gordon S. Blair

Computing Department
Lancaster University, UK
{s.lin,f.taiani,gordon}@comp.lancs.ac.uk

Abstract. Overlay networks have emerged as a powerful paradigm to realise a large range of distributed services. However, as the number of overlays grows and the systems that use them become more interconnected, overlays must increasingly *co-exist* within the same infrastructure. When this happens, overlays have to compete for limited resources, which causes negative interferences. This paper takes an opposite view, and argues that coexisting overlays may also introduce positive *synergies* that can be exploited to benefit a distributed system. Unfortunately, and in spite of some pioneering work, this phenomenon is still poorly understood and has yet to be investigated systematically. To address this problem, this paper proposes *a principled classification* of synergies, and illustrates how it can be used to exploit synergies in a typical overlay platform targeting gossip protocols (GossipKit). We review in detail the risks and benefits of each identified synergy; we present experimental data that validate their added value, and finally discuss the lessons we have learnt from our implementation.

Key words: coexistence, synergy, gossip, overlay framework

1 Introduction

An overlay network creates a virtual topology that is built on top of another virtual or physical topology. Over the past decade, overlay networks have emerged as a popular paradigm to offer more tailored services to specific classes of application such as multicasting, inter-domain routing, distributed file sharing and storage, and multimedia streaming [21, 9, 5, 6].

As more overlays are being developed, and as the systems that support them become more interconnected, overlays must increasingly *co-exist* in the same infrastructure. Because overlays are typically domain specific, a given node needs to support a growing number of overlays for different functions (e.g. a personal computer user might be making a Skype call over a VoIP overlay while downloading videos from a BitTorrent overlay). Furthermore, one overlay often depends on other overlays in intricate patterns, and must co-exist with them: for instance, T-Man [9] relies on overlays that maintain random graph to build

various structured network topologies. Finally, changing requirements in long-running systems will lead deployed overlays to be replaced, adding a dynamic dimension to the co-existence of overlays.

When overlays coexist in the same infrastructure, they may affect each other adversely: they must compete for node and network resources; they may also interact in unexpected ways, causing inconsistencies [4, 3]. In this paper we take an opposite view, and look instead at the cases when coexisting overlays might benefit from each other through potential *synergies* — i.e. when a set of mutually collaborating protocols perform more efficiently than the individual protocols operating in isolation.

Specific cases of synergies have been studied in a number of pioneering works [1, 5, 2]. Unfortunately, and in spite of these works, the general phenomenon of synergies remains poorly understood, and has never been studied systematically. This lack of analytical framework in turn limits the ability of overlay developers to analyse and exploit synergies, and prevent users of complex distributed systems to receive the full benefits that coexisting overlays can bring.

As a first step towards addressing these challenges, this paper presents a principled classification of synergies (Section 2), and illustrates its value by demonstrating how it can be used to identify and exploit synergies in GossipKit [8], a representative component framework that supports coexisting gossip overlays. More precisely, we discuss four concrete examples of synergies that demonstrate the main categories of our classification, and highlight their potential risks and benefits (Section 3). We then present experimental data based on a prototype implementation based on GossipKit that validate the added values of these synergies (Section 4). We finally discuss the lessons learnt from this work (Section 5), before discussing related work and concluding (Sections 6 and 7).

2 A Classification of Synergies Between Coexisting Overlays

We define synergy as the *beneficial emergent behaviour* of a set of *coexisting and collaborating overlays* that achieves a higher efficiency than when they operate in isolation. Synergies provide a *global* benefit through the *local* adaptation of each overlay's behaviour, thus delivering a high payoff at a minimum cost.

Since synergies result from the interactions of coexisting overlays, they depend on the *orientation* of these overlays in a system's physical architectural space, i.e. on how overlays coexist with respect to each other. In addition, and because of their nature, synergies must involve the collaboration of some key elements of each overlay. We have termed these key elements *facets*. *Orientation* and *facets* provide the two main dimensions of our classification (Fig. 1).

Communication protocols in general, and overlays in particular, are very often layered in stacks [14, 18]. In terms of *orientation*, this translates into two classic forms of coexistence (Fig 1): The *horizontal orientation* indicates overlays that execute in parallel at the same layer of a system, but do not normally require each other to deliver their individual service, in the same way that say TCP and

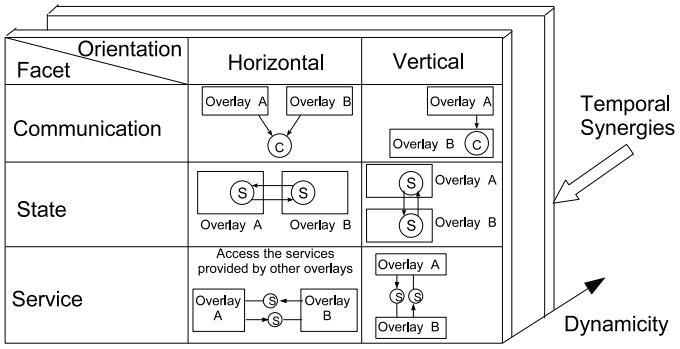

Fig. 1. A Classification of Synergies

UDP reside above IP. Meanwhile, the *vertical orientation* occurs when an overlay uses the functionality of an overlay located in a lower layer to provide its services. For instance, Ghodsi et al. [2] discuss how gossip-based unstructured overlays can be built on top of a structured overlay such as Chord [21].

In terms of *facets*, three key elements are likely to introduce synergies between overlays: their patterns of communication (*communication facet*), their states (*state facet*), and the possible external services they rely on (*service facet*). The *communication facet* refers to how the nodes of an overlay maintain a particular virtual topology by selecting neighbours according to specific control algorithms. When overlays coexist, their *communication facets* can often be coordinated locally to cause them to communicate with the same destination in a way that does not degrade the original service. When this happens, the network traffic can be reduced through piggybacking [22, 23] by merging together the data and control messages of multiple overlays to the same destination.

The *state facet* denotes the local data such as sensor readings, routing tables or unstable message buffers that an overlay maintains on a node. The *state facets* of coexisting overlays can often be shared to improve the overall performance of a distributed system through *state synergies* [5, 1]. For instance, the Synergy [5] networking platform enables co-existing overlays to share their local routing information. As a result, the messages of a given overlay can utilise the routes maintained by another overlay if those are shorter. This kind of cooperation is analogous to code sharing between airlines: a customer with a ticket from airline A might use airline B's flight on part of his journey if the two companies have agreed to share certain routes. Another example of state synergy is the joint overlay proposed by Maniymaran et al. [1]. In this work, the structured overlay Pastry [12] organises nodes in a ring topology. It coexists with an unstructured Clustering overlay that brings together nodes with close interests (e.g. similar videos or games on the node). When operating independently, both overlays need to maintain a list of virtual neighbours along with a "pool" of random peers. This pool acts as a shortcut for joining nodes to quickly discover their neighbours. Maniymaran et al. point out that the virtual neighbours of each overlay can be used as a pool of random peers by the other. Doing so significantly reduces the

network messages (2/3 of the overall messages) used by both overlays.

Finally, the *service facet* denotes the combination of distributed services that an overlay may provide. For instance, the prime functionality of a structured overlay such as Pastry is a standard key-based routing mechanism. However, Pastry's ring topology can also be used to broadcast messages or to aggregate data. Recognising this particular mix of services can allow an overlay that requires *both* routing and broadcast to use Pastry alone instead of a combination of two different overlays [2]. This in turn saves both computing and network resources. We refer to this kind of overlay "consolidation" as *service synergies*.

As mentioned in our introduction, long-running systems means overlays need to be reconfigurable at run-time. This dynamicity adds a third *temporal* dimension to the orientations and facets we have just discussed (Fig. 1). *Temporal synergies* refer to the situation when the initialisation of a newly loaded overlay can benefit from pre-existing overlays, in particular from those being replaced.

3 Case Study: Identifying Synergies within GossipKit

The classification of synergies that we have just presented provides a space of potential synergy types, which can serve as a guideline for developers to identify and exploit synergies between coexisting overlays. To evaluate this point, we apply here this classification to GossipKit [8], a representative component framework that supports coexisting gossip overlays, and discuss four synergy examples that illustrate the key categories of Figure 1.

3.1 Background: Gossip overlays and GossipKit

Gossip algorithms allow information to spread over a network in the way a rumour is randomly gossiped amongst a group of people. Gossip-based overlays have been widely applied to provide highly scalable communication in both IP-based networks [9, 10, 16, 17] and mobile adhoc environments [19]. To deliver scalable communication in a large fixed network, a gossip algorithm repeatedly exchanges limited data with a fixed number of randomly selected peers during gossip rounds. The random selection of peers can itself be implemented as a gossip overlay (e.g. RPS [10]). Gossip overlays in mobile adhoc networks differ in that they favour a probabilistic broadcast mechanism rather than a random selection of neighbours, and tend to be reactive rather than periodic, but their probabilistic and scalable approach remains unchanged [19].

Gossip overlays offer many classic examples of coexistence and hence are ideal candidates for the concrete study of synergies: they are often composed of simpler gossip overlays to achieve more complex tasks [11] (vertical coexistence); they are frequently employed to maintain various aspects of structured and unstructured overlays [9] (vertical/horizontal coexistence); and because they are built on randomised mechanisms and behave stochastically, are sometimes replicated to improve reliability (horizontal coexistence) [17].

GossipKit [8] is a fine-grained component framework that we have developed to ease the development of (re)configurable gossip overlays. In particular, it provides a toolkit to develop middleware platforms that support the coexistence of multiple gossip overlays. To illustrate its value, we have used GossipKit to implement eight different gossip-based overlays[1]. We have shown that GossipKit promotes code reuse, simplifies the configuration of gossip-based middleware, and supports the concurrent execution of multiple gossip overlays [8].

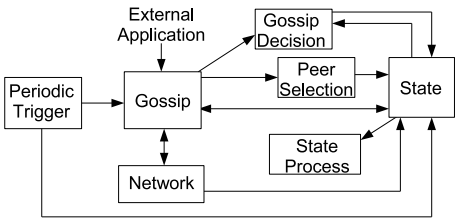

Fig. 2. GossipKit's Common Interaction Model

GossipKit's design follows a component model that captures the key elements of most gossip overlays [8] (Fig. 2). The `Gossip` component orchestrates the dissemination of data to random peers selected by the `Peer Selection` component via the `Network` component. The `Network` component supports inter-node interactions by encapsulating different communication mechanisms (TCP, UDP, or another overlays). The `Gossip` component can either be triggered reactively by events from external applications or periodically by the `Periodic Trigger` component. The `State` component maintains the data (e.g. a temperature reading, a list of temporary network packets or a set of neighbouring nodes) that is gossiped between nodes and is updated by the `State Process` component. Finally, the `Gossip Decision` component captures the conditions that trigger a gossip dissemination. This component model is recursive — each component can itself be implemented as a gossip overlay that follows the same component model. For instance, the `Peer Selection` component that provides topological maintenance can itself be a membership overlay such as RPS [10]. GossipKit provides different implementations for each of the above components, and thus allows users to realise a wide range of gossip-based overlays by simple composition.

3.2 Synergy Identification

To explore synergies within GossipKit, we first need to map each of three facets of our classification (Fig. 1) onto the components of GossipKit's model. In Figure 2, the `State` component directly matches the *state facet* in Figure 1. The `Peer Selection` component decides the communication target of individual

[1] The source code of these overlays and of GossipKit is available on line: `www.lancs.ac.uk/postgrad/lins6/GossipKit.html`

nodes, and is therefore a prime candidate for potential *communication* synergies. The `Network` component can itself be implemented as an overlay, and may thus provide additional services (*service facet*) that are exploitable by the gossip overlays. The remaining four components in Fig. 2 do not directly match any of our three overlay facets: they encapsulate processes that are limited to local computations, and are thus unlikely to provide exploitable synergies.

In the following we use this mapping to analyse four scenarios of overlay coexistence, both horizontal and vertical, and we investigate their potential for synergies through a study of the functionalities and semantics of the `State`, `Peer Selection` and `Network` components of each overlay.

3.2.1 Potential Horizontal Communication Synergy A *Communication Synergy* coordinates the *communication facets* of coexisting overlays so that these overlays can share the same set of communication targets and reduce the number of their network messages through piggybacking (see Section 2).

This situation may occur for instance when a distributed file storage system uses two gossip overlays for different parts of its functionalities: a *failure detection* overlay [16] to aggregate evidence of failed nodes; and *a data aggregation overlay* [17] to calculate the average data storage on overlay nodes and assist with load-balancing of the system. The impacts of a communication synergy between these two overlays can be analysed as follows.

Benefit (Network Usage): When operating independently, these two overlays must both select a constant number of M random peers to communicate with. The probability of a given node to be selected in a gossip round is $\frac{M}{N}$, and these two overlays will on average have $M \times \frac{M}{N} = \frac{M^2}{N}$ communication targets in common, thus contacting an overall average of $2M - \frac{M^2}{N}$ peers at each round. In large-scale networks ($N \gg M$), the number of messages sent during each round will thus approaches $2M$, and more generally the total number of messages will increase linearly with the number of coexisting overlays.

In contrast, using communication synergy, both overlays can consolidate their communication needs and invoke the `Peer Selection` component only once during each round. This allows both overlays to communicate with the same set of M random peers during the same round. With the exploitation of piggybacking mechanism, this approach limits the number of peers to be contacted to a constant M, regardless of the number of coexisting gossip overlays, and correspondingly reduces the number of network messages.

Risk (Aggregation Speed): Despite the above benefit, communication synergy raises a potential risk as the randomised exchange of information (e.g. through random peer selection) is crucial for a data aggregation gossip overlay to achieve a high convergence speed (typically $O(logN)$ rounds). *Communication Synergies* might therefore jeopardise an overlay's overall efficiency by introducing interferences in its peer selection process.

3.2.2 Potential Horizontal State Synergy A *State Synergy* considers that coexisting overlays can benefit from sharing the data they maintain. We have

already mentioned the work of Maniymaran et al. [1]. As a further example, consider the coexistence of T-MAN [9] and Clustering [1]. T-MAN constructs various logical topologies (e.g. a ring) from any initial random graph, and maintains a set of peers that are closer to the local node's coordinates (i.e. the *neighbour set* state). In addition to its *neighbour set* state, T-MAN also relies on the RPS protocol (see Section 3.1) to provide a *random* set of nodes that act as shortcuts for joining nodes to quickly discover their neighbours. Clustering (Section 2) is a gossip overlay that maintains nodes with close interests (e.g. similar videos or games on the node) in its *cluster* state. Similarly to T-MAN, Clustering also use RPS to obtain a *random* state. Because T-MAN uses random node coordinates, its *neighbour set* and Clustering's *cluster* state are not strongly correlated. One overlay's neighbour set can thus be viewed as random by the other overlay. By sharing their states, Clustering can use T-MAN's *neighbour set* as its *random* state and T-MAN can access Clustering's *cluster* set for the same purpose, thus eliminating the need to maintain the RPS overlay.

Benefit (Network Usage): Without state synergy, the combination of T-MAN, Clustering, and RPS generates a total number of $3 * M$ messages at each gossip round, where M is the number of peers contacted by each overlay. The state synergy eliminates RPS from the system, hence potentially reducing the total number of network messages to $2 * M$.

Risk (Convergence Speed): Both Clustering and T-MAN aim for convergence: T-MAN converges from any random topology to a predefined structure, while Clustering converges to an unstructured topology where nodes with close interests are linked. Both overlays eventually reach a stable situation, where the contents of T-MAM's *neighbour set* and Clustering's *cluster* state remain unchanged. As a result, when they approach convergence, each overlay might fail to provide enough random peers to the other, thus reducing the effectiveness of shortcuts, and slowing down both overlays' convergence speed. Therefore, to safely exploit this synergy type, developers must ensure that the state contents of two overlays are actually uncorrelated.

3.2.3 Potential Temporal State Synergy GossipKit supports the reconfiguration of gossip-based overlays at runtime, introducing the opportunity of temporal synergies (see Section 2). For instance, consider the case where the SCAMP overlay [15] is being replaced by T-MAN/RPS [9] because of changes in application requirements. SCAMP is a light-weight membership overlay that maintains a random partial membership with maximal entropy to ensure an optimal propagation of broadcast messages. Compared with SCAMP, T-MAN/RPS constructs various logical topologies from any initial random graph to provide a number of services that are not limited to broadcasting. It does so by updating each node's local membership view with the closest logical neighbours (i.e. as defined by a ranking function) on receipt of periodically exchanged random membership information. In order to bootstrap its topological construction, T-MAN normally relies on RPS. This bootstrapping stage requires each joining peer to contact some well-known starting peers, which then propagate join messages to

the rest of the overlay.

Benefit (Stabilisation Speed): Using state synergy, T-MAN/RPS can work more efficiently by reusing the random partial view previously maintained by SCAMP. Intuitively, SCAMP already contains a lists of nodes belonging to the system. By using this lists, RPS can speed up its joining process, and directly interact with multiple other group members, rather than a limited set of well-known peers, and thus stabilise more rapidly to a stable global state.

3.2.4 Potential Vertical Service Synergy As introduced in Section 3.1, the `Network` component of GossipKit can be implemented by an underlying overlay that provides a communication service (e.g. Pastry). Such overlays often exhibit properties in addition to communication (e.g. Pastry's ring topology). These properties can be leveraged to provide additional distributed services to the stacked overlays that use them, and thus *consolidate* services.

For instance, consider some gossip overlays that run atop a structured overlay such as Chord [21] to exploit its efficient broadcasting mechanism [2]. In this case, Chord can also provide a random peer sampling service, thus eliminating the need for an explicit sampling overlay such as RPS. This can be done by generating random identifiers from Chord's identifier space, followed by a distributed lookup with Chord's *find_successor* API [21].

Benefit (Flexibility): In contrast with gossip-based membership overlays that can only provide a fixed number of random peers in any gossip round, structured overlays can provide a set of random peers of any size.

Benefit (Network Usage): Chord's *find_successor* algorithm only requires a worst case of $\log N$ network messages to find a random peers in a network of N nodes, which is potentially much more efficient than the periodic $2 * N$ messages (2 indicates the bi-directional exchange of information) used by a gossip-based membership overlay such as RPS.

Risk (Network Usage): A periodic gossip overlay typically requires every node to select a random peer to gossip at each gossip round. Since each random peer selection takes $\log N$ messages, the global number of messages will reach $N * \log N$ per gossip round for a network of N nodes with Chord, which is less efficient than the $2 * N$ messages of RPS.

4 Quantitative Evaluation

The four synergies that we have just presented provide both potential benefits and potential risks. To validate their actual value, and assess how difficult they are to realise, we have implemented each of them in GossipKit. We first present below a quantitative evaluation based on this implementation and a network simulation. We will then move on (in Section 5) to discuss the lessons we learnt from this implementation.

In the experiments that follow, we used the network simulator Jist/SWANS [20] underneath GossipKit to provide a virtual network environment. In particular, this experimental set-up maintained GossipKit's ability to insert and remove overlays at run-time, thus allowing us to study temporal synergies.

4.1 Horizontal Communication Synergy (Failure Detection and Data Aggregation)

As explained in Section 3.2.1, the reduction in network messages that this synergy provides might come as the cost of an increased number of gossip rounds for the data aggregation of both protocols. To investigate this drawback, we perform the following experiment.

We execute both the failure detection overlay and the data aggregation overlay within GossipKit. In our implementation, both overlay protocols are configured to periodically send their local states (i.e. part of the global information that must be aggregated) to one randomly selected peer. A simulation run terminates when both overlays have converged — i.e. when the data aggregation overlay has reached a global average and the failure detection overlay has gathered liveness information about all the system's nodes. Each simulation run can be parameterised to exploit the communication synergy (i.e. by selecting the same random peer) or not. The corresponding simulation results, averaged over 20 simulation runs, are shown in Fig. 3 and Fig. 4 for various network sizes.

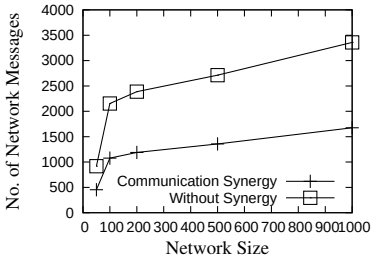

Fig. 3. Less network messages with the communication Synergy (ave. -50%)

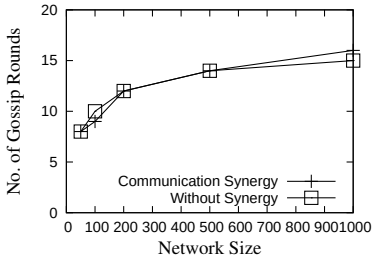

Fig. 4. Gossip rounds not degraded with the communication synergy

Benefit (Network Usage): Fig. 3 indicates that network messages are reduced by 50% when both overlays select the same peer at each gossip round and then merge their messages to be sent to this peer (piggybacking). This shows that communication synergy can significantly save network resources.

Risk (Aggregate Speed): Fig. 4 shows that the expected $O(logN)$ convergence rate remains as the network size grows, thus demonstrating that in this case the communication synergy does not affect the scalability of either overlays, and does not come at a cost of a slower convergence rate. This result can be generalised to any pair of independent gossip overlays who have identical randomness requirements.

4.2 Horizontal State Synergy (T-MAN and Clustering)

We have seen in Section 3.2.2 how some gossip overlays can share each other's state to obtain random links, and thus eliminate the need for an explicit peer sampling overlay such as RPS. To evaluate this type of synergy, we execute

T-MAN and Clustering in parallel (see Section 3.2.2), and run two types of simulations. In the first type, T-Man and Clustering rely on RPS to select random peers, while in the second, they share each other's state to obtain random links (state synergy).

Benefit (Network Usage): The number of messages required by both types of simulation, averaged over 20 runs, is shown in Fig. 5 for various network sizes. On average, the state synergy reduces network messages by 35% across all network sizes.

Risk (Convergence Speed): As mentioned in Section 3.2.2, T-MAN and Clustering might not be able to provide each other with peers that are "random enough" to maintain their speed of convergence. We look at this risk in Fig. 6 which plots the number of rounds needed for both T-MAN and Clustering to converge for various network sizes. This number remains unchanged with or without state synergy. The reasons for this is because the main topologies maintained by T-MAN and Clustering are not correlated. (T-Man maintains peers that are close in a virtual coordinate space while Clustering maintains peers who share similar interests.) Thus both overlays can provide enough random peers for each other even once they have stabilised. In fact, this reasoning applies to any pair of coexisting gossip overlays that maintain both a main topology and a random set of peers, as along as their main topologies are uncorrelated. Maniymaran's [1] joint use of Pastry and Clustering is a similar example of this type of state synergy (see Section 2).

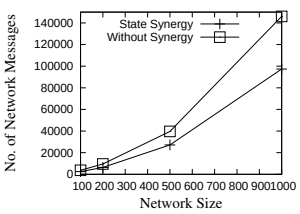

Fig. 5. Network messages are reduced with a state synergy (ave. -35%)

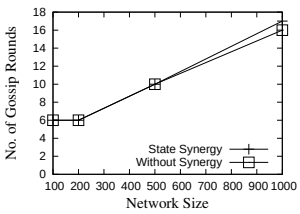

Fig. 6. Convergence remains unchanged with a state synergy

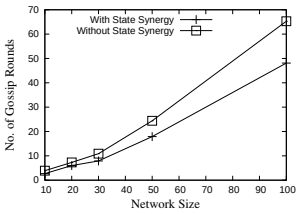

Fig. 7. Bootstrap speed improved with a temporal synergy (ave. -26%)

4.3 Temporal State Synergy (SCAMP Replaced by T-MAN)

In Section 3.2.3, we explained how nodes running T-MAN/RPS could potentially reduce their bootstrapping overhead by reusing the membership information maintained by a preexisting SCAMP overlay. The following experiment quantifies this reduction by using T-MAN/RPS to construct a ring topology in two sorts of simulation. In the first sort, T-MAN/RPS starts on each node with a empty local view (i.e. a bootstrap is required), while in the second, SCAMP is executed first and used to initialise T-MAN/RPS with a resulting set of random peers. We terminate both types of runs when T-MAN/RPS converges to a ring.

In all these experiments, T-MAN maintains a local state of size 2, and exchanges messages with both neighbours during every round; RPS maintains a local view of size 4 and communicates with only one random peer per round. SCAMP, as per construction, does not limit the size of its local view. Instead its sample size approximates $\log N$ for a network of N nodes. In the simulations in which T-MAN/RPS initialises its state with SCAMP's, RPS only keeps four random peers from SCAMP's view if SCAMP has more than four.

Benefit (Stabilisation Speed): Fig. 7 shows the number of gossip rounds needed by T-MAN to converge (averaged over 20 runs), with and without temporal state synergy. As expected, the synergy helps reduce the number of gossip rounds (an average of 26% reduction) needed to construct a ring topology on various network sizes.

4.4 Vertical Service Synergy (Gossiping Over Chord)

In Section 3.2.4, we have analysed that gossip overlays can exploit the *find_successor* service of a structured overlay such as Chord to select random peers on demand, and have discussed the potential risks of doing so for periodic gossip overlays. To evaluate the actual benefits and risks, we carry out the follow experiment with Chord configured to use an identifier space between 0 and 159.

Benefit (Flexibility): The RPS overlay can only return a limited number of random peers per round: calling the service repeatedly in the same round will return the same peers. In contrast, Chord can support an unlimited number of queries over any period of time. Fig. 8 illustrates this by plotting the distribution of 1000 random queries made by the same node on a Chord ring.

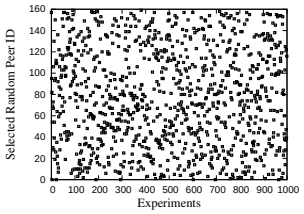

Fig. 8. 1000 Uniformly Random Peers are Selected for a Random Node on Chord

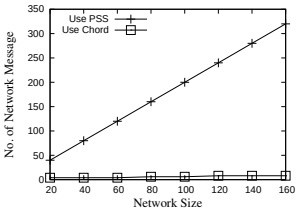

Fig. 9. The Network Usage for Selecting Random Peers on Demand

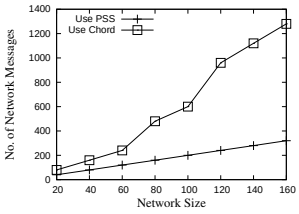

Fig. 10. The Network Usage for Selecting Random Peers Periodically on All Nodes

Benefit (Network Usage): Fig. 9 shows the average number of messages used by a single invocation of Chord's *find_successor*. The graph shows that Chord is indeed much more efficient than RPS if random peer selection is only needed occasionally, because it avoids the periodic flooding generated by RPS. Chord is thus a perfect candidate to maintain a static random graph, e.g. to broadcast probabilistic messages, and only occasionally select some random peers when a node joins or leaves the network (i.e. on demand).

Risk (Network Usage): However, our experiments (Fig. 10) have also confirmed that our case study of service synergy uses more network messages if it is used to support gossip overlays that require all nodes to gossip periodically.

5 Discussion

Fig. 11 summarises the results of the previous section for the synergies we have presented. In the remainder of this section, we discuss the lessons we learnt from this implementation and from our experimental evaluation.

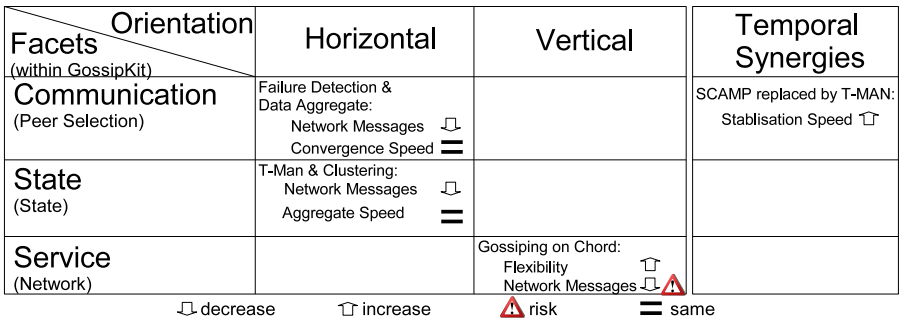

Fig. 11. A Summary of the Synergy Examples within GossipKit

1. We have demonstrated the existence of synergies for each of the four synergies we identified in Section 3.2, based on the classification we proposed in Section 2. The fact that GossipKit is a fine-grained component-based framework played a major role in this identification. This is because fine-grained component frameworks often clearly separate the key functional facets of an overlay, and hence help analyse these facets' potential for synergies. They also simplify the implementation of synergies by facilitating collaboration between facets.

2. Our *temporal synergy* example shows that synergies do not only exist between overlays executing in parallel, but also between overlays that coexist transiently during a reconfiguration. We think this directly impacts the design of future reconfigurable middleware, which should ideally support the automatic exploitation of synergies. Towards this aim, it seems that an overlay's state structure should be made explicitly manipulable by the middleware to support automatic reasoning.

3. To be exploitable, potential synergies should not violate the "good properties" of existing overlays. For instance, a main advantage of gossip overlays is their ability to converge in logarithmic rounds. Synergies, in particular those that limit the amount of entropy present in a platform, should maintain this.

4. Finally, while implementing the above four synergies, we found GossipKit's reflection mechanisms to be particularly helpful. For instance reflection allowed us to easily expose internal interfaces to other overlays. The use of a component model also opens the path for adaptor components that can resolve incompatible interfaces between different overlays.

6 Related Work

The iOverlay [6] framework was one of the earliest attempts to support overlay networks. It is essentially a low-level software cross-connect that forwards messages according to a script that embodies the semantics of a particular overlay. Macedon [7] provides a high-level domain specific language to facilitate the configuration of overlays. Both iOvelay and Macedon provide generic platforms for overlays, but do not focus on the coexistence of multiple overlays. GridKit [18] is a reflective middleware framework that supports the coexistence and cooperation of multiple of overlays in the same system. ODIN-S [4] addresses resource contention between coexisting overlays, and provides a scheduling mechanism to optimise resource usage. Very few studies have considered exploiting the potential synergies amongst coexisting overlays, and most of them only considered a particular instance of synergies. We have already discussed many of these works [5, 1, 2]. In addition, Ucan et al. [22] proposed a piggybacking mechanism to reduce overheads between multiple gossip broadcasts in a wireless sensor network.

7 Conclusion

In this paper, we have shown that coexisting overlays can introduce a wide range of potential synergies to benefit a distributed system. More precisely, as an early approach towards the systematic study of synergies, we have proposed a *principled classification* of synergies and demonstrated how this classification can help identify exploitable synergies within GossipKit, a representative component framework for gossip overlays. We have discussed the benefits and risks of each identified synergies, and provided an experimental evaluation on each of them, before discussing the lessons learnt from our experiments.

This paper opens up several exciting avenues of research. First, our *classification* could be further refined by studying a broader range of overlays. Second, because synergies are fraught with risks, an assessment mechanism seems inevitable to ensure that the exploitation of synergies does not cause negative side-effects to overlay systems. Finally, since new overlays will emerge and might need to be dynamically deployed into a distributed system without restart (e.g. a long-life system), we think that new (re)configuration mechanisms will be needed that directly support the dynamic exploitation of synergies.

Acknowledgement: This work has been partially supported by ESF MiNEMA Project and by the EU FP7 ICT Project WISEBED n. 224460.

References

1. B. Maniymaran, M. Bertier, A-M. Kermarrec. *Build One, Get One Free: Leveraging the Coexistence of Multiple P2P Overlay Networks.* In Proc. of 27th International Conference on Distributed Computing Systems, 2007.
2. A. Ghodsi, S. Haridi, H. Weatherspoon. *Exploiting the Synergy Between Gossiping and Structured Overlays.* In Proc. of ACM SIGOPS Op. Sys. Review, 2007.

3. S. Steinhauer, P. Okanda, G. Blair *Virtual Overlays: An Approach to the Management of Competing or Collaborating Overlay Structures.* In Proc. of 8th IFIP Conference on Distributed Applications and Interoperable Systems, 2008.
4. B. Cooper. *Trading Off Resources Between Overlapping Overlays.* In Proc. of Middleware Conference, 2006.
5. M. Kwon and S. Fahmy. *Synergy: An Overlay Internetworking Architecture.* In Proc. of 14th Inter. Conf. on Computer Communications and Networks, 2005.
6. B. Li, J. Guo, et al. *iOverlay: A Lightweight Middleware Infrastructure for Overlay Application Implementations.* In Proc. of IFIP/ACM Middleware Conf., 2004.
7. A. Rodriguez, C. Killian, C. Bhat, and et al. *MACEDON: Methodology for automatically creating, evaluating, and designing overlay networks.* In Proc. of USENIX/ACM Symposium on Networked Systems Design, 2004.
8. Shen Lin, Francois Taiani, Gordon Blair. *Facilitating Gossip Programming with the GossipKit Framework.* In Proc. of the 8th IFIP International Conference on Distributed Applications and Interoperable Systems, 2008.
9. M. Jelasity and O. Babaoglu, *T-Man: Gossip-based overlay topology management.* In EngineeringSelf-Organising Systems: 3rd International Workshop, 2005.
10. M. Jelasity, R. Guerraoui, A. Kermarrec et al. *The Peer Sampling Service: Experimental Evaluation of Unstructured Gossip-Based Implementations.* In Proc. of the 5th ACM/IFIP/USENIX international conference on Middleware, 2004.
11. E. Riviere, R. Baldoni, H. Li, et al. , *Compositional gossip: a conceptual architecture for designing gossip-based applications.* ACM SIGOPS Op. Sys. Review, 2007.
12. A. Rowstron and P. Druschel *Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.* IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), 2001.
13. A-M. Kermarrec, L. Massoulie, A. Ganesh, et al. *Probabilistic Reliable Dissemination in Large-Scale Systems.* IEEE Trans. Parallel Distrib. Syst. 2003.
14. K. Birman, A. Abbadi, W. Dietrich, et, al. *An Overview of the ISIS Project.* IEEE Distributed Processing Technical Committee Newsletter. January 1985.
15. A. Ganesh, A.-M. Kermarrec and L. Massoulie, *SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication.* In Proc. of the 3rd International workshop on Networked Group Communication, 2001.
16. R. Renesse, Y. Minsky and M. Hayden, *A gossip-style failure-detection service.* In Proc. Distributed Systems Platform and Open Distributed Processing, 1998.
17. M. Jelasity, A. Montresor and O. Babaoglu, *Gossip-based aggregation in large dynamic networks.* ACM Trans. Comput. Syst. 23, 3 (Aug. 2005), 219-252.
18. P. Grace, G. Coulson, G. Blair et al. *GRIDKIT: Pluggable Overlay Networks for Grid Computing.* Proc. Int. Symp. on Distributed Objects and Applications, 2004.
19. R. Friedman, D. Gavidia, L. Rodirgues et al. *Gossiping on MANETs: the Beauty and the Beast.* ACM Operating Systems Review, 2007.
20. R. Barr, Z. Haas and R. van Renesse, *JiST: an efficient approach to simulation using virtual machines: Research Articles.* Software Practical Experiments, 2005.
21. I. Stoica, R. Morris, D. Liben-Nowell, and et al., *Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications.* In Proc. of 2001 ACM SIGCOMM.
22. Ercan Ucan, Nathanael Thompson, Indranil Gupta. *A Piggybacking Approach to Reduce Overhead in Sensor Network Gossiping.* In 2nd International Workshop on Middleware for Sensor Networks (MidSens'07).
23. V. Ananthanarayana and K. Vidyasankar. *Dynamic Primary Copy with Piggy-Backing Mechanism for Replicated UDDI Registry.* In Proc. of Distributed Computing and Internet Technology, 2006.