Signaling and Reciprocity: Robust Decentralized Information Flows in Social, Communication, and Computer Networks

THÈSE N^{O} 4961 (2011)

PRÉSENTÉE LE 25 FÉVRIER 2011

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
LABORATOIRE DE SYSTÈMES RÉPARTIS
PROGRAMME DOCTORAL EN INFORMATIQUE, COMMUNICATIONS ET INFORMATION

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Wojciech GALUBA

acceptée sur proposition du jury:

Prof. B. Faltings, président du jury
Prof. K. Aberer, directeur de thèse
Prof. J.-P. Hubaux, rapporteur
Dr B. A. Huberman, rapporteur
Dr W. Kellerer, rapporteur

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Suisse
2011

Contents

Abstract

Complex networks exist for a number of purposes. The neural, metabolic and food networks ensure our survival, while the social, economic, transportation and communication networks allow us to prosper. Independently of the purposes and particularities of the physical embodiment of the networks, one of their fundamental functions is the delivery of information from one part of the network to another. Gossip and diseases diffuse in the social networks, electrochemical signals propagate in the neural networks and data packets travel in the Internet.

Engineering networks for robust information flows is a challenging task. First, the mechanism through which the network forms and changes its topology needs to be defined. Second, within a given topology, the information must be routed to the appropriate recipients. Third, both the network formation and the routing mechanisms need to be robust against a wide spectrum of failures and adversaries. Fourth, the network formation, routing and failure recovery must operate under the resource constraints, either intrinsic or extrinsic to the network. Finally, the autonomously operating parts of the network must be incentivized to contribute their resources to facilitate the information flows.

This thesis tackles the above challenges within the context of several types of networks: 1) *peer-to-peer overlays* - computers interconnected over the Internet to form an overlay in which participants provide various services to one another, 2) *mobile ad-hoc networks* - mobile nodes distributed in physical space communicating wirelessly with the goal of delivering data from one part of the network to another, 3) *file-sharing networks* - networks whose participants interconnect over the Internet to exchange files, 4) *social networks* - humans disseminating and consuming information through the network of social relationships.

The thesis makes several contributions. Firstly, we propose a general algorithm, which given a set of nodes embedded in an arbitrary metric space, interconnects them into a network that efficiently routes information. We apply the algorithm to the peer-to-peer overlays and experimentally demonstrate its high performance, scalability as well as resilience to continuous peer arrivals and departures.

We then shift our focus to the problem of the reliability of routing in the peer-to-peer overlays. Each overlay peer has limited resources and when they are exhausted this ultimately leads to delayed or lost overlay messages. All the solutions addressing this problem rely on message redundancy, which significantly increases the resource costs of fault-tolerance. We propose a bandwidth-efficient single-path Forward Feedback Protocol (FFP) for overlay message routing in which successfully delivered messages are followed by a feedback signal to reinforce the routing paths. Internet testbed evaluation shows that FFP uses 2-5 times less network bandwidth than the existing protocols relying on message redundancy, while achieving comparable fault-tolerance levels under a variety of failure scenarios.

While the Forward Feedback Protocol is robust to message loss and delays, it is vulnerable to malicious message injection. We address this and other security problems by proposing

Castor, a variant of FFP for mobile ad-hoc networks (MANETs). In Castor, we use the same general mechanism as in FFP; each time a message is routed, the routing path is either enforced or weakened by the feedback signal depending on whether the routing succeeded or not. However, unlike FFP, Castor employs cryptographic mechanisms for ensuring the integrity and authenticity of the messages. We compare Castor to four other MANET routing protocols. Despite Castor's simplicity, it achieves up to 40% higher packet delivery rates than the other protocols and recovers at least twice as fast as the other protocols in a wide range of attacks and failure scenarios.

Both of our protocols, FFP and Castor, rely on simple signaling to improve the routing robustness in peer-to-peer and mobile ad-hoc networks. Given the success of the signaling mechanism in shaping the information flows in these two types of networks, we examine if signaling plays a similar crucial role in the on-line social networks. We characterize the propagation of URLs in the social network of Twitter. The data analysis uncovers several statistical regularities in the user activity, the social graph, the structure of the URL cascades as well as the communication and signaling dynamics. Based on these results, we propose a propagation model that accurately predicts which users are likely to mention which URLs. We outline a number of applications where the social network information flow modelling would be crucial: content ranking and filtering, viral marketing and spam detection.

Finally, we consider the problem of freeriding in peer-to-peer file-sharing applications, when users can download data from others, but never reciprocate by uploading. To address the problem, we propose a variant of the BitTorrent system in which two peers are only allowed to connect if their owners know one another in the real world. When the users know which other users their BitTorrent client connects to, they are more likely to cooperate. The social network becomes the content distribution network and the freeriding problem is solved by leveraging the social norms and reciprocity to stabilize cooperation rather than relying on technological means. Our extensive simulation shows that the social network topology is an efficient and scalable content distribution medium, while at the same time provides robustness to freeriding.

Keywords: complex networks, mobile ad-hoc networks, social networks, peer-to-peer systems, routing, fault-tolerance, information diffusion

Rèsumè

Les réseaux complexes existent dans de multiples domaines. Les réseaux de neurones, les réseaux métaboliques et les réseaux de nourriture assurent notre survie, tandis que les réseaux sociaux, économiques, de transport et de communication nous permettent de prospérer. Indépendamment de l'application et des particularités du mode de réalisation physique de ces réseaux, l'une de leurs fonctions fondamentales est la livraison d'informations d'une partie du réseau à l'autre. Les commérages et les maladies se diffusent dans les réseaux sociaux, les signaux électrochimiques se propagent dans les réseaux de neurones et les paquets de données se transmettent dans l'Internet.

Développer des réseaux pour des flux d'information robustes est une tâche difficile. Tout d'abord, le mécanisme par lequel le réseau se forme et en change la topologie doit être défini. Deuxièmement, au sein d'une topologie donnée, l'information doit être acheminée vers les destinataires appropriés. Troisièmement, tant la formation des réseaux que les mécanismes de routage doivent être robustes contre un large spectre de problèmes et d'adversaires. Quatrièmement, la formation du réseau, le routage et la récupération après une défaillance doit fonctionner indépendamment des contraintes au niveau des ressources, qu'elles soient intrinsèques ou extrinsèques au réseau. Enfin, les parties du réseau agissant de manière autonome doivent être incitées à partager leurs ressources afin faciliter la circulation de l'information.

Cette thèse aborde les défis ci-dessus dans le cadre de plusieurs types de réseaux: 1) *les réseaux pair à pair* - des ordinateurs interconnectés via Internet formant un réseau dans lequel les participants fournissent divers services les uns aux autres, 2) *les réseaux mobiles ad-hoc* - des nœuds mobiles distribués physiquement dans l'espace communiquant sans fil dans le but de fournir des données d'une partie du réseau à l'autre, 3) *les réseaux de partage de fichiers* - réseaux dont les participants interconnectés via Internet s'échangent des fichiers, 4) *les réseaux sociaux* - des êtres humains diffusent et consomment des informations à travers le réseau des relations sociales.

La thèse apporte plusieurs contributions. Tout d'abord, nous proposons un algorithme général, qui, étant donné un ensemble de noeuds intégrés dans un espace métrique arbitraire, interconnecte ces noeuds dans un réseau qui route les informations efficacement. Nous appliquons cet algorithme dans le contexte des réseaux pair à pair et démontrons expérimentalement ses performances élevées, son évolutivité ainsi que sa résistance face aux incessantes arrivées et départs de pairs.

Nous nous concentrons ensuite sur le problème de la fiabilité du routage dans un réseau pair à pair. Chaque pair possède des ressources limitées et l'épuisement de ces dernières conduit à des retards de livraison des messages ou à leur perte. Toutes les solutions à ce problème impliquent une redondance des message, ce qui augmente considérablement les coûts des ressources nécessaires à la tolérance aux pannes. Nous proposons un protocole de routage, nommé *Forward Feedback Protocol (FFP)*, économique en bande passante, dans lequel les messages correctement délivrés sont suivis par un signal de retour utilisé pour renforcer les routes choisies. Les évaluations menées dans un banc d'essai sur Internet indiquent que FFP utilise de 2 à 5 fois moins de bande passante que les protocoles existants s'appuyant sur la redondance des messages, tout en offrant des niveaux de tolérance aux pannes comparables pour une variété de scénarios de panne.

Bien que le protocole soit robuste face à la perte de messages et aux retards, il est vulnérable à l'injection de messages malveillants. Nous abordons également d'autres problèmes de sécurité en proposant *Castor*, une variante de FFP pour les réseaux mobiles ad-hoc (MANET). Dans Castor, nous utilisons le même mécanisme général que pour FFP: chaque fois qu'un message est routé, le chemin de routage est soit renforcé ou affaibli par le signal de retour, selon que l'acheminement ait réussi ou non. Cependant, contrairement à FFP, Castor emploie des mécanismes cryptographiques pour assurer l'intégrité et l'authenticité des messages. Nous comparons Castor à quatre autres protocoles de routage MANET. Malgré la simplicité de Castor, il délivre jusqu'à 40% plus de paquets sur une période donnée que les autres protocoles et récupère au moins deux fois plus vite que les autres protocoles dans un large éventail d'attaques et de scénarios de panne.

Nos deux protocoles, FFP et Castor, s'appuyent sur la signalisation simple pour améliorer la robustesse de routage dans les réseaux pair à pair et les réseaux mobiles ad-hoc. Compte tenu du succès du mécanisme de signalisation relatif à la définition du flux d'information dans ces deux types de réseaux, nous examinons si la signalisation joue un rôle critique dans les réseaux sociaux en ligne. Nous caractérisons la propagation des URL dans le réseau social Twitter. L'analyse des données révèle plusieurs régularités statistiques dans l'activité de l'utilisateur, le graphe social, la structure des cascades d'URL ainsi que dans les communications et la dynamique de signalisation. Sur la base de ces résultats, nous proposons un modèle de propagation qui prédit avec précision les utilisateurs qui sont susceptibles de mentionner ces URL. Nous présentons un certain nombre d'applications où la modélisation des flux d'information du réseau social serait cruciale: le classement du contenu et de filtrage, le marketing viral et la détection de spam.

Enfin, nous considérons la problématique dite du "profiteur" (ou "freeriding") dans les applications pair à pair de partage de fichiers, lorsque les utilisateurs peuvent télécharger des données depuis d'autres utilisateurs, mais ne rendent jamais la pareille. Pour résoudre le problème, nous proposons une variante du système BitTorrent dans laquelle deux pairs ne sont autorisés à communiquer que si leurs propriétaires se connaissent dans le monde réel. Lorsque les utilisateurs savent quels sont les clients BitTorrent qui se connectent à eux, ces derniers sont plus susceptibles de coopérer. Le réseau social devient un réseau de distribution de contenu et le problème du profiteur est résolu en tirant parti des normes sociales et de la réciprocité. Notre simulation montre que la topologie du réseau social est un efficace et évolutif moyen de distribution du contenu, tout en étant robuste face aux profiteurs.

Mots clefs: réseaux complexes, les réseaux mobiles ad-hoc, réseaux sociaux, réseaux pair à pair, routage, tolérance de panne, diffusion d'information

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my thesis advisor Prof. Karl Aberer for his guidance, continuous in-depth feedback, our enlightening and often perspective-changing discussions and for the creative freedom I have been given during the work on the thesis.

Secondly, I would like to acknowledge Prof. Jean-Pierre Hubaux, Dr. Bernardo A. Huberman, Dr. Wolfgang Kellerer and Prof. Boi Faltings for generously agreeing to serve on my thesis committee and providing valuable comments and suggestions on my work.

My gratitude also goes to all my EPFL research collaborators without whom some of the chapters of this thesis could not have come to be. My sincere appreciation goes to Dipanjan Chakraborty, Panos Papadimitratos and Marcin Poturalski for contributing their valuable expertise and sharing their inexhaustible passion for problem solving.

I am also immeasurably indebted to Wolfgang Kellerer and Zoran Despotovic for hosting me on my internship at Docomo Euro-Labs, which both personally and professionally has been a truly enriching experience. It is impossible to understate Wolfgang's and Zoran's key role in guiding the research in the right direction all throughout the work on this thesis.

I would also like to extend my utmost gratitude to Bernardo A. Huberman for giving me the opportunity to spend the summer at the Social Computing Lab, HP Labs. It has been an incredible learning experience, which has brought in important new perspectives into both my research and my worldview. Many thanks to Daniel M. Romero and Sitaram Asur for the creative brainstorming sessions and the hard work we have done together as well as for mercilessly obliterating me in table tennis.

The work on ProtoPeer, a key par of the thesis, would not have been completed without the early prototyping done by Julien Herzen and Jean Respen. I am most grateful for their perseverance in fighting the uphill battle of engineering large-scale distributed systems. I would also like to thank Frank Lehrieder for his numerous contributions to the project, in particular, the network models, which opened up new research directions in this thesis.

A big thank you goes to all of my friends, lab mates and fellow grad students for sharing the same Swiss boat with me all these years, for the mutual inspiration with new ideas and for reminding each other that there is life beside the thesis. A very special thanks goes to the DOSErs et al. for the wonderful time spent on the road and in the Alps, for the steady reliable stream of fun and for coping with the almost non-differentiable elevation profiles of the bike trips that you joined me on.

Finally and most of all, I would like to thank my parents for their enduring love, patience and support on this long voyage. Mamo, Tato, dziękuję!

Chapter 1

Introduction

Networks are everywhere around us. Take a simple act of Alice having a cell phone conversation on the train. When Alice speaks with Bob, the microphone turns the sound waves into electrical signals that are then processed by a network of highly integrated circuits inside the phone and emitted through the antenna as electromagnetic radiation. That signal travels to a cell phone network tower appropriately chosen based on the current location of Alice's train. The tower turns the signal into data, which is then forwarded through several communication networks using different media and protocols and is then received by Bob's phone and emitted to him as audible sound waves. The vibrations of Bob's eardrum become the electrical signals that are fed into Bob's brain. Powered by a complex network of metabolic pathways, Bob's neural network processes the information and the airwaves are interpreted as Alice's voice. Finally, the semantic networks of concepts underlying the human language allow Bob to formulate a meaningful response to Alice.

Both Alice and Bob are part of larger networks. They are lucky to be mostly on the receiving end of the multi-species food chain supporting them with nutrients. They are part of the human social network, which is the medium for the spread of ideas, innovation as well as diseases and behaviors. The train that Alice is riding is part of the larger transportation network, without which the long supply chains necessary to manufacture Alice's cell phone could not exist. Neither would the cell phone exist without the complex economic networks of contracts, companies and markets.

All these networks - biological, technological and socioeconomic - vary greatly in their structural complexity, scale, dynamics and purpose. In the past two decades, there has been an unprecedented growth in the amount of available data about these networks as well as a vast improvement in processing capabilities needed to analyze the gathered data. This has led to the flourishing of the science of complex networks.

1.1. The science of complex networks

A network, in its most basic graph-theoretic sense is a set of nodes interconnected with edges or arcs. This might be an overly reductionistic simplification, but such a general definition enables the application of the same set of analytical and statistical frameworks to the study of a wide range of networks. One of the more striking results coming out of the complex networks research is that despite the fact that the networks vary widely in their purpose and function, they share a large number of commonalities both in their structure as well as the processes ocurring on them. The analysis of a range of biological, technological and socioeconomic networks has shown that their structure follows certain universal rules: networks exhibit power-law distributions in the number of nodes each node connects to, the path lengths between any pair of nodes are short and there is a high likelihood that two nodes might have common neighbors. Many *network formation mechanisms* have been proposed that would reproduce the observed structural properties (§2.1.1).

The structure of the networks is often tied to their function and the processes that occur on top of them. In particular, a large number of these networks serve as a medium for propagation of data, physical objects, node state or other entities. In many cases, the flow or diffusion in these networks is the very purpose of their existence.

1.1.1. Information Flows

Epidemiology studies the spread of diseases in the networks of human contact. Sociologists look at how ideas and behaviors are copied from one person to another in the social networks. Neuroscientists model the propagation of electrochemical signals in the neural networks. Telecommunication engineers build networks to reliably and quickly deliver data from one location to another. All these processes can be broadly defined as *information flows* in which some piece of information moves from one node to another. Looking at these processes in such a general way has several advantages.

Firstly, flow modeling can be done independently of the entities that are flowing in the network and independently of the medium facilitating the flow. Secondly, the relationship between the structure of the network and the properties of the information flow can be studied in a general way producing insights valid across many types of networks. Finally, mechanisms designed for one type of network can be successfully applied in other types of networks.

This point of view is key to this thesis. We take the general view on the information flows in networks, explore the problem space and propose solutions that are universally applicable to a wide range of networks or operate at the intersection of different types of networks.

1.1.2. Routing

The information in networks normally does not indiscriminately propagate from any node to any other node, but rather selectively only along specific links. For example, diseases spread only to the nodes susceptible to infection, neurons fire only if they have been sufficiently stimulated by their neighbors. Humans tell others a story if they personally think it is interesting. Packets on the Internet are most of the time forwarded only to one specific router in each hop.

The selection of the links along which the information flows is either intentional: a packet needs to be forwarded towards a specific IP address, or opportunistic: a virus only spreads through susceptible hosts. Each network has its own *routing machanism* by which the direction of the information flow is chosen.

The information cannot directly flow between nodes that are not connected. The routing mechanism must thus operate within the constraints of the network topology. The network topology, in turn, is determined by another mechanism that controls how the network forms and changes over time. Consequently, there are two fundamental mechanisms shaping the information flows in complex networks: network formation and routing. The decomposition of network operation into these two mechanisms is a fundamental step taken in this thesis that enables us to talk about the information flow engineering in general terms in the context of several different types of networks.

1.1.3. Scalability & robustness

At small scales, all networks can be created and controlled by a centralized entity. Take, for example, a local cable company in a small town. The decisions on how to interconnect all the houses are made by a single engineering team. There is enough manpower to handle the network servicing and connecting new customers.

At larger scales, networks tend to be composed of autonomous components, each independently forming and maintaining the network topology as well as autonomously deciding on how to route information. The Internet and the global transportation system are prime examples of such networks. Subdivision into largely independent parts is one of the key factors enabling the scalability of complex networks in terms of their size and capacity. In the extreme case, the autonomy might be at the level of a single node. In the social networks, each node makes independent decisions about which other nodes to connect to and communicate with. In the neural networks, each neuron, depending on the local electro-chemical conditions autonomously forms new connections and routes information.

Irrespective of the organization of the networks and the levels of autonomy, each network needs to be robust against failures. A network failure is any departure in node operation from their evolved or designed function. For example, a misconfigured Internet router might stop correctly forwarding packets, the transportation network might be disrupted by volcano ash, a switch failure might cause a domino effect of failures in the power grid ultimately leading to a larger regional outage, spontaneous local neuronal synchronization might escalate and cause an epileptic fit.

Whether it is the customer base or the organism's survival that are at stake, networks must be either designed or evolved to be robust against failures. Modeling the failures and understanding their impact on the network operation is absolutely crucial. In its most basic form, a failure is simply a node or a link ceasing to function, often randomly and independently of other nodes or links. Partly because of the simplicity of such failures, a significant part of the complex networks research is devoted to studying how networks respond to them.

There is good understanding on how the removal of edges or nodes from the network affects their connectivity and the diffusion processes ocurring on them. However, in reality, failures are often more complex than random independent node removals. This is particularly true if the failures are intentional. A network might, for example, be under an attack whose goal is to create a large disruption in the network's operation while controlling only a small number of network nodes. The intentionality is typically captured by an adversary model that clearly states the goals of the attacker as well as the constraints under which the attacker is allowed to operate. The network's vulnerability to the attacks is a function of not only the attacker model, but also the network topology, the network's signaling and communication protocols and the attack countermeasures that are put in place. It is thus hard to reason about fault tolerance and security without going into the specifics of the networks. Designing robust information flow mechanisms that would be universally applicable to different types of networks is a challenging problem that we take on in this thesis.

1.1.4. Resource constraints & cooperation

Networks composed of autonomous parts create an entirely new set of robustness problems. While the subdivision into independent components allows the networks to scale, the autonomous parts must cooperate to ensure the correct functioning of the network as a whole. But cooperation is not a given. There frequently exists a conflict between the interests of the network and the interests of its individual components. For example, a node in a peer-to-peer file-sharing system might be downloading files, but not, as it is expected of it, upload files to others. In another network, the Internet, the top-level architecture is composed of Autonomous Systems (ASes). To get delivered, the data needs to be forwarded across several ASes. In order to achieve that, the ASes must cooperate by forwarding one another's traffic, but at the same time they must set aside enough resources to serve their own clients.

At the core of the cooperation problem are the limited resources available in the networks. In the file-sharing networks, there is limited upload and download bandwidth. Sensor networks must operate under stringent energy constraints. In the transportation networks, there are limits on road throughput, airport capacity and train frequency. In the social networks, there is scarcity of human attention that can be devoted to consuming or forwarding the information.

Given all these constraints, the autonomously operating network parts might selfishly conserve resources for their own benefit, while having little incentive in expending them for the good of the other network parts. The conflict between the goals of the parts and the goals of the whole is one of the central design problems of building large-scale robust networks.

1.1.5. Engineering robust information flows

Complex networks exist for a number of purposes: communication, transportation, cognition, survival, etc. Independently of the purposes, and particularities of the physical embodiment of the networks, there frequently is a need to deliver information from one part of the network to another. Engineering networks for robust information flows is a challenging task. Let us restate the problems we have identified so far:

- The mechanism through which the network forms and changes its topology needs to be defined.
- Within a given topology, the information must be routed to the appropriate reciepients.
- Both the network formation and the routing mechanisms need to be robust against a wide spectrum of failures and adversaries.
- Network formation, routing and failure recovery must operate under the resource constraints, either intrinsic or extrinsic to the network.
- The autonomously operating parts of the network must be incentivised to contribute their resources to facilitate the information flows.

This thesis tackles the above challenges within the context of several networks:

- **peer-to-peer overlays** - computers interconnected over the Internet to form an overlay in which participants provide various services to one another
- **mobile ad-hoc networks** - mobile nodes distributed in physical space communicating wirelessly with the goal of delivering data from one part of the network to another
- **social networks** - humans disseminating and consuming information through the network of social relationships
- **file-sharing networks** - networks whose participants interconnect over the Internet to exchange files

1.2. Outline & contributions

In Chapter 2 we briefly cover the fundamentals and the related work pertaining to each of the areas addressed in this thesis. The background chapter is followed by a series of technical chapters, the outlines and contributions of which can be found below. The thesis concludes in Chapter 8, which is a synthesis of all the findings in the thesis, the relationships between them and the outlook for the future directions of research. This is followed by the Appendix A that describes the distributed systems simulation and live deployment toolkit that was developed for the needs of the thesis and used throughout it. The toolkit itself is a significant contribution to systems evaluation research and is publicly available at `http://protopeer.net`.

1.2.1. Flexible peer-to-peer overlay network formation

Routing in most complex networks has been recently hypothesised to be greedy and to rely on hidden metric spaces of node attributes to make progress towards the destinations [1]. An interesting question that follows from this hypothesis is whether there exists a general algorithm capable of constructing a network and routing in it that would work with nodes embedded in arbitrary metric spaces. In Chapter 3, we answer positively to that question in the context of peer-to-peer overlay networks.

Peer-to-peer (P2P) systems allow for harnessing vast distributed resources for file-sharing, video streaming or computation (§2.2). In almost all applications, peers form a *peer-to-peer overlay*, an ad-hoc network that allows the peers to locate and communicate with each other. The P2P overlay needs to have a topology that supports efficient information flows, but that at the same time requires few resources for structural maintenance as peers join and leave the overlay and as failures happen in the system. We propose an overlay routing and maintenance algorithm that for peers placed in an abstract metric peer identifier space is able to wire the peers into an efficiently routing and scalable topology. Our approach improves over the previous ones in several aspects. Firstly, in the state-of-the art, the peer identifier space is imposed by the specifics of the particular overlay structure. In our case, the application can define its own identifier space and identifier distance function, which offers great flexibility in application design. Secondly, unlike the existing approaches, our algorithm does not explicitly and actively maintain any topological invariants, but new links are created on-demand based on message traffic. The global topology is emergent rather than explicitly constructed. We demonstrate experimentally that in the overlays running our algorithm, the average routing path lengths and average node degrees increase logarithmically in terms of the network size. This scalability property is evaluated in simulation for a wide range of identifier spaces and under continuous peer arrivals and departures.

1.2.2. Fault-tolerant bandwidth-efficient overlay routing

Peer-to-peer overlays are usually large-scale and heterogeneous with the availability of local resources at each peer significantly varying across the network. If any of the local network, processing or storage resources are exhausted, this ultimately leads to delayed or lost overlay messages. The overlay routing algorithms have to be robust against this. We consider this problem in Chapter 4.

All the current approaches rely on message retries or multi-path routing to ensure message delivery under intermittent peer failures or message delays. Message redundancy significantly increases the resource costs of fault-tolerance. We propose a bandwidth-efficient single-path Forward Feedback Protocol (FFP) for overlay message routing in which each successfully delivered message is followed by a feedback signal on its routing path. By observing the signals, each peer locally learns which of its next hops are reliable for forwarding to which part of the overlay. The network as a whole converges on efficient loop-free routes and dynamically responds to failures by routing messages around the faulty peers.

We deploy our protocol together with several other existing approaches on a planetary-scale Internet testbed. The measurements show that FFP uses 2-5 times less network bandwidth than the existing protocols relying on message redundancy, while at the same time achieving comparable fault-tolerance levels under a variety of failure scenarios.

1.2.3. Secure and scalable routing in mobile ad-hoc networks

While the Forward Feedback Protocol is robust to message loss and delays, it is vulnerable to malicious message injection. More specifically, the attacker could send fabricated positive feedback messages convincing the nodes downstream on the routing path that the delivery is successful, while at the same time dropping the messages that are being routed. We address this and other security problems in Chapter 5 by proposing Castor, a variant of FFP for mobile ad-hoc networks (MANETs).

Like the peer-to-peer overlays, MANETs are highly dynamic networks, the nodes can join or leave at any time and the network topology continuously changes as the wirelessly communicating mobile nodes move in and out of each other's radio ranges. The dynamic and ad-hoc nature of these networks leads to a number of system design tradeoffs between efficiency, scalability and security.

In Castor, in contrast to FFP, instead of "forward feedback" there is "backward feedback" in the form of acknowledgments returning back from the destination after the correct delivery of each data packet. The acknowledgment, similarly to the case of FFP's feedback messages, serves as the path reinforcement signal. As in FFP, that signal is used by each node on the path to keep track of the routing reliability of its neighbors. However, unlike FFP, all messages sent in Castor contain additional cryptographic material that allows all the nodes on the path to verify the origin and the integrity of the received messages. We compare Castor to four other MANET routing protocols under a wide range of attacks, network scales and node mobility scenarios. Despite Castor's simplicity, it achieves up to 40% higher packet delivery rates than the other protocols, while having similar or smaller bandwidth overhead. In Castor, nodes rely only on their own local observations and never exchange any routing state with the other nodes. This allows the system to rapidly react to local changes in the topology or packet loss caused by the attacks. In our measurements, Castor recovers at least twice as fast as the other protocols in a wide range of failure scenarios.

1.2.4. Information flows in on-line social networks

The FFP and Castor protocols proposed in the preceding chapters use simple signaling to reinforce reliable paths in the network and as a result, the whole network collectively becomes more fault-tolerant and routes more efficiently. Given the success of the signaling mechanism in shaping the information flows in these the peer-to-peer and mobile ad-hoc networks, we next turn to examining if signaling plays a similar crucial role in on-line social networks.

In Chapter 6, we characterize the propagation of URLs in the social network of Twitter, a popular microblogging site, where users exchange short messages. When a user likes the other user's message, she “retweets” it to her friends, which also lets the original poster know that someone liked their message. We examine how this basic signaling mechanism influences the information flows in the social network.

We track 15 million URLs exchanged among 2.7 million users over a 300 hour period. Data analysis uncovers several statistical regularities in the user activity, the social graph, the structure of the URL cascades and the communication dynamics. Based on these results we propose a propagation model that predicts which users are likely to mention which URLs. The model correctly accounts for more than half of the URL mentions in our data set, while maintaining a false positive rate lower than 15%. We outline a number of applications where the ability to accurately predict the information spread in on-line social networks would be crucial: content ranking and filtering, viral marketing and spam detection.

1.2.5. Social networks as content distribution networks

In Chapter 7 we consider the problem of cooperation in peer-to-peer file-sharing applications. In the popular file-sharing network, BitTorrent, peers discover one another via a centralized server and essentially form an efficient random content distribution topology. Participation in the system requires that peers devote a substantial amount of bandwidth to uploading to other peers, mostly strangers, which creates an incentive to freeride, i.e. only download but not upload any content to others. Users may also choose not to upload for privacy or liability reasons. To address these problems, we propose a variant of the BitTorrent system in which two peers are only allowed to connect if their owners know one another in the real world. When the users know which other users their BitTorrent client connects to, they are more likely to cooperate. With this architectural change, the social network becomes the content distribution network and the freeriding problem is solved by leveraging the social norms and trust to stabilize cooperation rather than relying on technological means.

Our extensive simulation of the social BitTorrent system produces several insights. First, the social network topology alone without the central servers is an efficient and scalable content distribution medium. Second, thanks to the cooperative social links, BitTorrent's robustness to freeriding significantly improves. Finally, we find that a hybrid solution in which peers download from both their friends and other peers obtained from the central servers has the highest robustness to freeriding, shortest download completion times and the most balanced upload bandwidth utilization.

Chapter 2

Background

2.1. Complex networks

In the past two decades, there has been a dramatic rise in the availability of data about many types of networks as well as the increase in the processing capabilities available for analyzing it. This has led to the flourishing of the science of complex networks. Unlike graph theory [2], which focuses on theoretical studies of regular, abstract networks, the interest of complex networks research lies in the real-world, large-scale, highly irregular and dynamically evolving networks. There have been many surveys [3, 4, 5] and books [6] published in the area and the reader is referred to them for a more comprehensive overview. In the remainder of this section, we only briefly highlight the work that is most relevant to the thesis.

2.1.1. Network structure and formation

During the initial surge in data availability, much of the research in the area of complex networks focused on their topology. One of the most prevalent characteristics of complex networks is their small-world property, that is, the fact that relative to the size of the network it takes only a few steps to reach one node from another by traversing the network links. Moreover, the probability that two nodes have a common neighbor is higher than if it was happening by chance. The small-world property was empirically confirmed in a wide variety of networks: starting from the social networks and the seminal Milgram experiment [7], through networks of co-authorship [8], telephone call graphs [9], the Internet [10], World Wide Web [11] and neural networks [12].

Although, at first, the consistency of the small-world property across the different types of networks might be a surprising fact, on closer inspection it turns out to be a consequence of another commonly occurring topological pattern. If for most nodes, the number of nodes reachable within r hops is increasing exponentially with r then the average shortest path length between the nodes will be proportional to the logarithm of the number of nodes in the graph, i.e., the graph will be a small-world.

The small-world property is easily achievable by graph rewiring. This has been demonstrated by Watts and Strogatz [13]. Their model starts with graph nodes positioned on a low-dimensional regular lattice. The nodes are then connected to other nodes that are k or fewer nodes away on the lattice, creating a mesh of local interconnectivity. Then for each edge with probability p, one of its ends is rewired to another randomly chosen node. This process leads to a small-world graph, which at the same time has a high clustering coefficient, a property that is frequently observed in real networks together with the short average path length.

The small-world property has broad implications for the diffusion processes occurring in complex networks and the fact that it is relatively easy to achieve by simple rewiring of the network is essential to designing networks that support efficient information flows.

Apart from the small-world property, complex networks have several other commonly occurring structural characteristics: power-law distribution of the node degree, high clustering coefficient and high degree correlation of connected nodes [5]. Several models of network formation beside the already mentioned Watts-Strogatz model [13] have been proposed.

Perhaps the simplest model is the random graph model first studied by Rappaport [14] and Erdös and Rýeni [15]. In that model, an n node graph is constructed by deciding for each of the $\frac{1}{2}n(n-1)$ edges whether it should exist or not with probability p. Such graphs have a binomial degree distribution and a short average path length. However, real-world networks are less uniform, they exhibit non-trivial structural patterns that suggest other mechanisms of network formation might be at work.

One of the most widely studied network growth processes is the *preferential attachment*. It was first proposed by Price [16] and then studied further by Barabasi and Albert [17]. In this model, a newly joining node connects to m other nodes, by choosing them with the probability proportional to their current degree. This model was the first one to successfully account for the power-law degree distributions ubiquitously observed in real-world networks.

Defining the network formation process is an essential first step in information flow engineering. The complex networks literature offers a wealth of guidelines for the design of the network maintenance and construction processes in computer and communication networks, but does not define the concrete message passing protocols that would drive these processes. This is the focus of Chapter 3 and is further discussed here in the context of peer-to-peer overlays (§2.2).

2.1.2. Diffusion processes

Knowing the structure of complex networks is not sufficient to completely understand their function. What largely determines the function are the processes that occur on top of the network structure. Of particular relevance to this thesis and receiving significant attention in the literature are the processes of diffusion occurring in the networks.

One of the central questions is how the structure of the networks affects the dynamics of the diffusion. This problem has been studied particularly well in the context of epidemics [18], where better understanding of diffusion can lead to improvement in disease prevention and vaccine development.

In the SIR model of epidemic spreading [19], the population is divided into three distinct classes: “S” - susceptible, “I” - infected and “R” - recovered/removed. A susceptible node becomes infected when in contact with another infected node. When the infection subsides or the node dies, it enters the “R” state from which it never returns. All state transitions and node contacts are probabilistic and given the initial state, the steady state can be solved for exactly.

The initial results on the SIR model unrealistically assumed perfect mixing between the nodes, i.e., there was no particular network structure through which the infections are occurring. Since then, these results have been expanded to describe epidemic processes on general networks [20]. One of the main results coming out of this work is the observation that as the probability of infection between the nodes increases, most networks undergo a rapid phase transition from the state where a small fraction of nodes is infected to a state in which most nodes are infected. The value of the infection probability at which this occurs is termed the *epidemic threshold* and computing this value is the main focus of modeling.

Because of the prevalence of networks with the power-law node degree distribution, social networks in particular, it is an important question of how that specific topology affects the epidemic spread. It has been shown that if the power-law coefficient is less than 3, such networks never have a non-zero epidemic threshold [21]. This means that independently of the probability of viral transmission either all diseases spread to most nodes or to a small fraction of nodes. Key to the spread in these networks are nodes of high degree. Following a random path in a power-law network has a high chance of reaching a high degree node. This has led to the development of vaccination practices that target highly connected individuals [22].

While the spread of the diseases is something that needs to be prevented, there are many diffusion processes with the opposite goal. Viral marketing campaigns in on-line social networks want to spread their message to as many people as possible [23], security updates in a peer-to-peer systems need to propagate as quickly as possible to many computers [24] and gossiping and anti-entropy approaches are frequently used in distributed systems to synchronize state across the system [25] or compute state aggregates [26]. These diffusion processes bear many analogies to the disease spread and in many cases, tools and concepts borrowed from epidemiology are applied to their analysis and design.

In particular, in this thesis we study two diffusion processes ocurring on the social networks. In Chapter 7 we look at how diffusion driven by the BitTorrent protocol allows the files to spread efficiently in the social network. The dynamics of diffusion in this case are mainly driven by the available upload bandwidth at each node and how it allocates it to each of its neighbors in the social network. In Chapter 6 we propose an accurate model that predicts the spread of information in the on-line social network of Twitter. The diffusion in this case is driven by the virality of the information and the influence that users have over one another.

2.1.3. Search and navigability

In the late 60's the sociologist Stanley Milgram was interested in mapping the structure of the social acquaintances. Due to the lack of large databases of social contacts at that time, Milgram conceived an experiment [7] that would map the social network pathways. Random individuals from the United States were each handed a copy of a letter addressed to the same target person. The participants of the experiment, when they received the letter, were asked to route it to another person that in their judgment would most likely know the target recipient. The experiment produced a surprising and widely cited result: on average the letter needed only 6 hops to reach the target. The social network was confirmed to indeed be a small-world.

Perhaps, the even more astonishing conclusion that can be drawn from Milgram's results is that people were able to efficiently route the letter without the global knowledge of the network. Their decisions were based only on the local set of acquaintances that they were aware of. Watts et al. [27] offer the following explanation of this phenomenon. Each individual has an identity - a set of characteristics measured along several dimensions. How different those characteristics are determines the social distance between two individuals. The model assumes that the chain letter is forwarded to the next hop acquaintance whose distance from the target is the lowest. Greedy routing in social networks based on social and geographical attributes was demonstrated to have both high delivery success rate as well as short average paths.

To determine exactly what topological properties a graph should satisfy in order to support efficient greedy routing, Kleinberg [28] puts the nodes on a multi-dimensional lattice. In each hop, the nodes attempt to minimize the lattice distance to the target. Each node is connected to its closest lattice neighbors, but also has one long-range link pointing to a distant node. The lengths of the long-range links in terms of lattice distance are drawn from a power-law distribution. The core finding of this work is that there exists an optimal value of the exponent of the distance distribution under which greedy routing becomes efficient and takes only $O(\log^2(N))$ hops in terms of the network size.

Greedy routing's reliance on only the local information and its efficiency have led to the hypothesis that it is likely to be the universal routing mechanism in most complex networks. Boguñá et al. [1] posit that nodes in complex networks are embedded in hidden metric spaces and demonstrate that the topologies of several real-world networks support efficient search by greedy routing.

In Chapter 3 we build on these results and design a network formation protocol capable of scalably constructing efficient topologies that support greedy routing in arbitrary metric spaces.

2.2. Peer-to-peer systems

A *peer-to-peer system (P2P)* consists of nodes (peers) communicating through some medium, typically the Internet [29, 30]. All peers are equal in their function and responsibilities. The main application of peer-to-peer systems is the efficient and scalable sharing of peers' resources:

storage, network bandwidth or processing power. Storage is shared in applications such as information retrieval (peer-to-peer web search), publish/subscribe services and the Semantic Web. However, by far the most popular application is the file-sharing, e.g., BitTorrent [31]. Bandwidth-sharing peer-to-peer systems are mainly aimed at the efficient streaming of data over the network. The systems's ability to find several disjoint paths from the source to the destination can significantly boost the streaming performance. Bandwidth-sharing peer-to-peer systems are mostly found in peer-to-peer telephony, peer-to-peer video/TV, sensor networks and peer-to-peer publish/subscribe services. For computationally intensive tasks, when the CPU resources of a single peer cannot fulfill its needs, the CPU-sharing peer-to-peer systems can provide plenty of CPU resources available at the idle overlay peers. Currently, several scientific experiments harness the peer-to-peer technology for simulation of protein folding[1] or analysis of an astronomic radio signals[2].

2.2.1. Overlay architectures

Peers in the P2P systems are usually interconnected into an overlay network. A peer joining the system discovers the other peers through some bootstrap protocol and then opens long-lasting connections to selected peers. to participate in the overlay topology. The fundamental functionality that all peer-to-peer overlays provide is the ability to locate the other peers or resources not by their IP addresses, but by their logical identifiers.

The peer-to-peer overlay architectures have undergone several redesigns over the years. The beginning of the file-sharing era and the rise and fall of the first partially centralized peer-to-peer file-sharing systems (Napster [32]) in the early 2000s paved the way for the second generation of peer-to-peer overlays (Gnutella [33], Freenet [34]). The simple protocols and the unstructured, decentralized nature made these networks robust and lacking the drawbacks of the centralized approaches.

It had soon become evident that the ad-hoc architecture of Gnutella-like systems does not scale and uses excessive amounts of network bandwidth. More efficient, structured overlays appeared (P-Grid [35], Chord [36], Kademlia [37]), which used the existing resources more effectively.

In *unstructured* overlays such as Gnutella [33] there are no constraints on how the topology can form as peers arrive and depart from the system. The peers that other peers can connect to are either discovered through random walks or through flooding. The discovery process is biased towards nodes which are already connected to many peers, which results in a highly unbalanced topology with a power-law degree distribution [38].

Unlike their unstructured counterparts, the topology of the *structured* overlays is not arbitrary. Link establishment among the peers is usually strictly defined by specific protocols. The topologies can result in various regular structures like rings [36], tries [35], multidimensional

[1] `http://folding.stanford.edu/`

[2] `http://setiathome.berkeley.edu/`

tori [39], de Bruijn graphs [40] or more loose randomized small-worlds [41].

2.2.2. Overlay routing

Peer-to-peer overlay networks enable the peers to communicate with one another even if the communicating peers do not know their addresses in the underlying network, e.g., the IP address on the Internet.The way it is achieved in the overlays is by *routing* overlay messages. Each overlay message originates at a source and is forwarded by the peers in the overlay until the message reaches one or more destinations. A number of routing schemes have been proposed.

2.2.2.1. Routing in unstructured overlays

Unstructured overlay networks use mainly two mechanisms to deliver routed messages: flooding and random walks (Fig. 2.1). When some peer v receives a flood message from one of its overlay neighbors w, then v forwards the flood message to all of its neighbors except w. When v receives the same flood message again, it is ignored. Eventually the flood reaches all of the destinations.

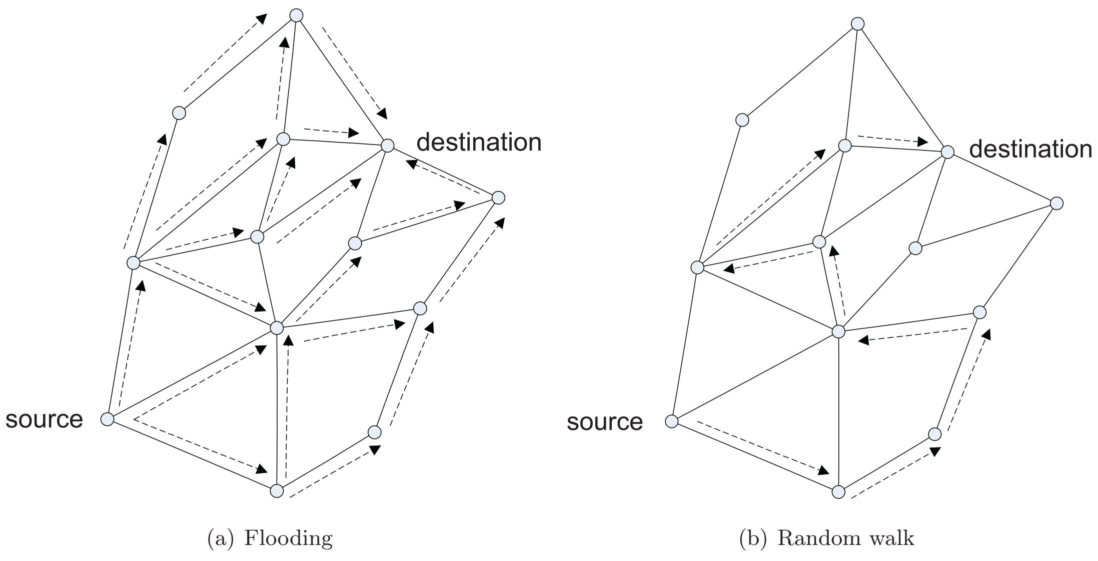
For example, in Gnutella [33], a file-sharing peer-to-peer system, a peer s that wants to download a file floods the network with queries. If some peer d that has the file desired by s is reached by the query flood, then d sends a response back to s. Flooding consumes a significant amount of network bandwidth. To reduce it, the flooded messages typically contain a Time-To-Live (TTL) counter included in every message. The counter is decremented whenever the message is forwarded. This limits how far the flood can spread from the source but at the same time lowers the probability of reaching the peer that holds the searched file.

The high bandwidth usage of flooding has led to the design of an alternative routing scheme for unstructured overlay networks: *random walks* [42]. Instead of forwarding a message to all of the neighbors, it is only forwarded to a randomly chosen one. Depending on the network topology random walks provide different guarantees of locating the destination peer(s), however all of the random walk approaches share one disadvantage: a significant and, in most cases, intolerable latency.

2.2.2.2. Routing in structured overlays

As more peers join the overlay network and as there are more messages that need to be routed, flooding and random walks quickly reach their scalability limits. This problem has prompted the research on structured overlays.

In structured overlays, each peer has a unique and unchanging identifier picked when a peer joins the overlay. The peer identifiers enable efficient routing in the structured overlays. Each routed message has a destination identifier to which the message needs to be delivered. Instead of blindly forwarding the message to all neighbors, as in the unstructured overlays, each peer in a structured overlay forwards the message to only one neighbor in each hop. The next hop is selected by using the concept of *peer identifier distance*.



(a) Flooding (b) Random walk

Figure 2.1. Routing in unstructured overlay networks. The circles and solid lines represent the overlay topology. The dashed arrows illustrate the flow of messages. Peers routing in an unstructured network do not know the exact location of the destinations so they have to either look in all possible directions via flooding or randomly walk to find the destination peer.

Most of the modern structured overlays define the notion of distance between any two peer identifiers. For example, in Chord [36] identifiers are selected from the set of integers $[0, 2^m - 1]$ and are ordered in a modulo 2^m circle. The distance $d(x, y)$ between two identifiers x and y is defined as the difference between x and y on that identifier circle, i.e., $d(x, y) = (y - x) \bmod 2^m$ In another overlay, Kademlia [37], the identifiers are 160-bit integers and the distance between two identifiers x and y is defined as their exclusive bitwise OR (XOR) interpreted as an integer, i.e., $d(x, y) = x \oplus y$.

Although the modern structured overlays differ in the details of how they make use of the peer identifiers for routing, they are all based on the same general *greedy routing* principle. When some peer v receives a message with a given destination identifier it forwards the message to that next hop whose identifier is the closest to the destination identifier. In other words, in every hop the message gets as close as possible to the destination. Routing terminates when Time-To-Live of the message is exhausted or the peer is the message destination. The topology of most structured overlays is designed to be scalable, that is, all destinations can be reached from any node in a logarithmic number of hops in terms of the number of hops.

2.2.3. Maintenance

Peer-to-peer systems are commonly deployed in highly dynamic environments, peers can depart or join the system at any time. These continuous joins and departures are commonly referred to as *churn*. Instead of gracefully departing from the network, peers can also abruptly fail or

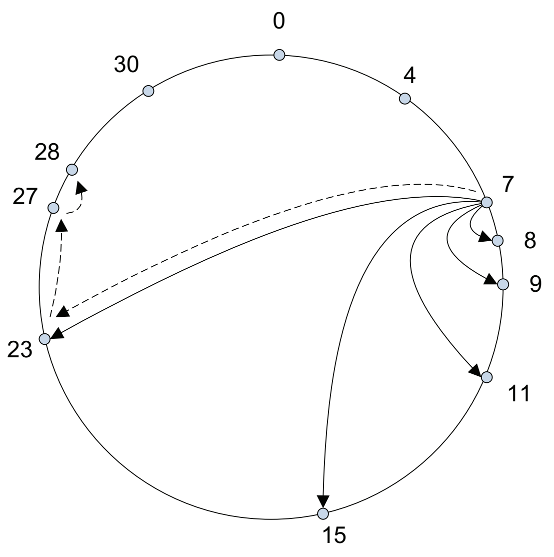


Figure 2.2. Routing in Chord. The big circle represents the peer identifier space with IDs in the interval $[0, 2^5]$. The small circles are the peers and the number beside them is their ID. Peer 7 is connected (solid arrows) to peers with exponentially increasing distance from 7: $7 + 1 = 8$, $7 + 2 = 9$, $7 + 4 = 11$, $7 + 8 = 15$, $7 + 16 = 23$. Assume that peer 7 wants to route a message to peer 28. Dashed arrows represent the routing path. The peer 23 is the neighbor of 7 that is closest on the ring to the destination 28 and that neighbor is chosen as the first hop for the message. One hop is not enough to reach 28, but the peer 23 brings the message closer to the destination and peer 27 finally delivers it. The greedy routing rule of always selecting such next hop that brings the message as close as possible to its destination is the main building block of all structured overlay networks.

the network connection with some of its neighbors may be closed. In all of these cases, the lost connectivity may adversely affect the performance of the system. The overlay topology needs to be continuously *maintained* to guarantee message delivery and routing efficiency.

There are two main approaches to overlay maintenance: proactive and reactive. In *proactive maintenance* peers periodically update their routing tables such that they satisfy the overlay topology invariants. For example, Chord [36] periodically runs a "stabilization" protocol to ensure that every peer is linked to other peers at exponentially increasing distance. This ensures routing efficiency. To guarantee message delivery, each Chord peer maintains connections to its immediate predecessor and successor on the Chord ring. Maintaining these ring connections ensures that a message can always make progress towards the destination even in the absence of the longer distance connectivity.

In contrast to proactive maintenance, *reactive maintenance* is triggered immediately after the detection of a peer failure or peer departure. The missing entry in the routing table is replaced with a new one by sending a connect request to an appropriate peer.

Failures and departures of peers are detected in two ways: by probing or through usage. In probe-based failure detection, each peer continuously runs a ping-response protocol with each of its neighbors. When ping timeouts occur repeatedly the neighbor is considered to be down and is removed from the routing table. In usage-based failure detection when a message is sent to a neighbor but not acknowledged within a timeout, the neighbor is considered to have failed.

The more neighbors a peer must maintain, the higher the bandwidth overhead incurred by the maintenance protocol. In modern structured overlays, the per-node maintenance bandwidth overhead typically scales as $O(\log(n))$ in terms of the network size.

2.2.4. Failure modes & security

Peer-to-peer overlays are composed of an ad-hoc collection of nodes forming a communication substrate for the message flows. For such a system to be reliable, each node must serve its intended function and correctly behave according to the prescribed protocols. In the simplest failure mode, a node or a link might simply stop functioning and responding to any communication attempts. Most systems are designed to handle this type of failures very well. However, nodes might deviate from their expected behavior in less trivial ways. This opens up a whole space of different failure modes. In particular, the cases where a larger fraction of nodes can fail simultaneously. Apart from benign failures, the network function can also be intentionally disrupted. Making networks robust against such intentionally introduced misbehavior is the subject of peer-to-peer security research.

Peer-to-peer systems are large-scale and have few restrictions on node joining, which creates a number of security vulnerabilities [43, 44]. By inserting many adversary-controlled nodes into the overlay or injecting spurious messages at high rate, an adversary might mount a Denial-of-Service (DoS) attack on the network to exhaust resources on a set of nodes (CPU, bandwidth, memory etc.) or target a specific set of nodes to prevent them from forwarding messages or lower the availability of a particular service or data that the nodes provide [45]. This problem is primarily addressed by techniques based on redundancy. Data and services are placed on multiple replicas and the messages are routed redundantly along several disjoint paths [46] with the goal that at least one of them will not pass through the nodes that are currently under the DoS attack.

An adversary may also place many nodes that it controls in a small part of the node identifier , effectively taking it over and preventing all nodes residing there from being accessed. This is known as the *Eclipse attack* and can be defended against by using statistical tests to detect and eliminate high degree nodes, which are more likely to be observed when the attack is taking place [47].

The Eclipse attack relies on the attacker being able to choose the node identifiers for the controlled nodes. In a more general version of this attack, the *Sybil attack* [48], the adversary inserts a large number of controlled nodes into the system severely disrupting its operation. Both the Eclipse attack and the Sybil attack can be mounted in peer-to-peer overlays with a relatively low cost in resources as thousands of overlay peers can operate on a single physical node. The main reason why this attack is so effective is the lax admission control to the overlay. Several solution have been proposed to address this [49]. The most common solution and, in fact, conjectured to be the only effective one [48] is the assignment of certified identities by a trusted authority.

A common application of peer-to-peer systems is data storage (§2.2). The data storage layer has its own set of vulnerabilities, attacks and countermeasures. The problem space includes: preventing content poisoning [50], access control [51] or ensuring censorship resistance [52]. However, we leave these problems out of the scope of the thesis.

In Chapter 4 we focus on the failure mode in which peers can either permanently or transiently drop or delay messages either for benign or malicious reasons. We design a routing protocol that can handle such failures, but unlike the state-of-the-art, our protocol does not rely on message redundancy thus providing many-fold savings in bandwidth.

2.3. Mobile ad-hoc networks

Mobile ad-hoc networks (MANETs), also known as mesh networks [53] consist of mobile nodes communicating wirelessly with one another. Unlike the modern cellular networks, in MANETS, there are no immobile base stations that facilitate communication between the mobile nodes, all nodes are usually equal in functionality and responsibility, which gives these networks substantial deployment and tear-down flexibility. Among the applications envisioned for such networks are: emergency search-and-rescue operations, data acquisition in hostile environments and ad-hoc communication in areas with limited infrastructure (e.g., the One Laptop per Child project[3]).

The main MANET design challenges stem from the high dynamicity and ad-hoc nature of these networks. Efficient and reliable communication needs to be sustained as the nodes join, leave and continuously change their location (§2.3.1). The wireless communication medium and the fact that new nodes can join the network at any time additionally bring in a set of challenging security problems (§2.3.2). All these problems have to be solved under a number of constraints. Firstly, the mobile nodes are typically battery powered and must carefully manage their energy. Secondly, the wireless medium is a shared resource and the bandwidth is limited. The communication protocols must minimize the control traffic overhead and maximize the data throughput. Finally, the radio transmission power and consequently the communication range is limited and the nodes need to cooperate to deliver the data over longer distances.

2.3.1. Routing

One of the most fundamental capabilities of MANETs is to enable the data communication between any two remote nodes even if they are not in their respective wireless communication

[3] `http://en.wikipedia.org/wiki/One_Laptop_per_Child`

ranges. The positions of the nodes in space and their radio capabilities determine the network's topology. When two nodes that are not directly connected need to communicate, they rely on the other nodes to forward the data between them. Thus, each node acts as a router and when it receives the data packet, it needs to decide which of its neighbors to forward it to. Routing needs to be efficient and robust as the nodes change their location, join or leave the network or undergo failures.

Routing protocol design is one of the fundamental research problems in mobile ad-hoc networks. Many protocols have been proposed and can be classified in different ways [54, 55]. Most of the protocols consists of two distinct processes: *route discovery* and *routing*. The role of route discovery is to find suitable routes for forwarding data from the given source to a given destination. Those routes are then used during routing.

Depending on how the route discovery operates, the protocols can be divided into two broad classes: on-demand and proactive. In on-demand protocols, route discovery is triggered only when there is a need to route to some specific destination and the routing information is not up-to-date. The most common protocol pattern for on-demand route discovery is a of route request originating at the source and being flooded throughout the whole network. When the route request reaches the destination, it sends a route response which propagates back to the source and establishes the route. Route discovery is also triggered, when due to topology changes some node determines that it is not possible to continue routing with the routing information the node currently has.

On the other hand, in proactive protocols, a continuously running route discovery process keeps the nodes up-to-date and prepared for routing to any destination. Nodes periodically exchange routes and the information about the state of the network topology. This comes at a bandwidth cost, and networks running proactive protocols need to have frequent and diverse enough data traffic to justify the expense of continuously keeping the routing information up-to-date. Otherwise, for more sporadic traffic, the on-demand routing protocols are the more suitable option.

The routing protocols can also be classified according to where the information necessary to make the routing decisions is kept. The two classes are: the *source routing* protocols and the *distance-vector* protocols. In the source routing protocols, the source aggregates all the routing information from the route discovery process. The source then includes the complete route in each data packet and the route is then used by the intermediate nodes for picking the appropriate next hop. In the distance-vector protocols, on the other hand, each node locally maintains a routing table, which for each destination stores the next hop that should be taken. Various protocols also store additional meta-data for each route, such as the distance to the destination measured in the number of hops, packet loss rates, sequence numbers to handle routing table update concurrency etc. Both in source routing and in distance-vector approaches, the routing information has to be kept fresh as the network topology and network conditions are changing. This can happen either through on-demand or proactive route discovery.

In another class of approaches, position-based routing [56], there is no route discovery, but the nodes instead rely on knowing the current location of their neighbors and the destination to make routing decisions. However, such protocols are out of scope of the thesis and we do not describe them here.

2.3.2. Failure modes & security

When the distance between two MANET nodes exceeds their communication range, the link between them fails. Moreover, the nodes may abruptly and permanently cease functioning due to hardware errors, energy depletion or a simple departure from the network. Those node and link failure modes are well handled by the MANET routing protocols (§2.3.1). However, just as in the case of peer-to-peer overlays, there are many other ways in which MANETs can fail either due to benign or malicious causes.

The goals of the adversary when attacking the MANET routing protocols (§2.3.1) may include [55]: taking control of part of the traffic between a subset of nodes, disrupting the network communication and degrade the quality of service or increasing the resource consumption on some nodes (energy, CPU, memory, etc.). The means available to the attacker to achieve these goals are numerous.

Possibly the simplest attack an adversary can perform is to jam the radio signals and effectively prevent any communication between nodes in some area. More sophisticated attacks can be mounted when the adversary has control over several nodes in the network. The adversary might then inject, replay, drop or modify messages to either prevent route discovery or force the routes to pass through the adversarial nodes. By sending fabricated messages, the adversary might also create corrupt routing state at the nodes to direct traffic into routing loops or non-existent links, which results in dropped packets. A particularly trivial to implement and effective is the grayhole attack in which the adversarial nodes correctly participate in route discovery, but subsequently drop all the data traffic that passes through them.

One of the commonly used countermeasures against these attacks is message origin authentication. The creator of the message attaches additional cryptographic material that can then be used by the recipients to verify the message origin. Similarly, aggregate signatures can be used to provide a proof that a message has indeed passed through all the nodes on the routing path, which is an essential step in securing the route discovery. A particularly challenging security problem is the protection of mutable message fields and ensuring that they are modified in the ways conformant to the protocols or ways that do not give advantage to the adversary.

In summary, there is unfortunately no silver bullet that can address all the security problems. Each protocol requires a separate security analysis and the design of highly specific mechanisms to counter the attacks. Moreover, the design of secure routing protocols for MANETs often faces trade-offs between security, performance and scalability. In Chapter 5 we propose a novel approach in which the routing and route discovery processes are merged together for increased robustness and security, while maintaining high protocol scalability.

2.4. Cooperation

Networks composed of autonomously operating nodes inadvertently run into the freeriding problem, when the nodes use the resources of the network, but do not contribute their own resources for the good of the network (§1.1.4). Virtually all the networks operate under some resource constraints, e.g., the bandwidth in the communication networks, the road capacity in the transportation networks or the limited human ability to consume information in the on-line social networks. The more severe the constraints the bigger the need for building or enforcing cooperation between the autonomous nodes. Among the most effective mechanisms for doing that are the trust, reputation and incentive systems.

2.4.1. Fundamentals

Assume a set of *nodes* continuously engaging in bilateral *interactions*. For simplicity, we assume that a single interaction always involves a pair of nodes and that interactions involving a larger group of nodes can always be decomposed into a set of binary interactions.

Each interaction has an associated *benefit* and *cost* for both nodes. These two values are normally such that nodes face a Prisoner's Dilemma (PD) [57], i.e., it is beneficial for the node to cooperate only if the other node cooperates as well, otherwise it is better not to cooperate.

When Alice interacts with Bob she can gain more if she is able to predict that Bob will cooperate. The extent to which a node believes the other will cooperate is the extent to which a node *trusts* the other node [58, 59]. There are a number of ways this belief can be inferred and they are captured by the different *trust models* [60]. One of the inferences that can be made is: if Alice cooperated with Bob then it implies Alice is also likely to cooperate with Carol. If this inference is applied universally, the collective actions of Alice form a commonly shared belief among the other nodes of how likely Alice is to cooperate. This belief shared among the nodes about some node is what is termed *reputation*. Reputation serves two roles. Firstly, it is a *public signal* that a node is worth cooperating with and secondly, it is a *sanctioning mechanism* through which the nodes can punish the other nodes for non-cooperation with bad reputation.

Another class of mechanisms for ensuring cooperation are the *currency-based* approaches, which assume the existence of some form of currency (tokens, scrip money, stamps etc.). When Alice cooperates with Bob, he pays her for the service. The payment can later be redeemed by Alice for cooperative behavior from another player. The players have the incentive to cooperate and accumulate the currency, otherwise they run the risk of not being able to afford to have other nodes cooperate with them when needed.

We next discuss the reputation and currency-based approaches in more detail.

2.4.2. Reputation-based systems

In reputation-based systems, each node determines trust towards other nodes based on the interactions it had with them. Every node V_i only has information about the actions of A that

V_i itself experienced. To compute the reputation of A, nodes need to exchange the information about the actions of A that they have observed. This exchange and the subsequent computation of reputation can proceed in many ways.

There are four classes of approaches [61]: trust graphs, probabilistic estimation, game-theoretic models and evolutionary approaches.

2.4.2.1. Trust graphs

The trust graph approach assumes the existence of a digraph of links between the nodes. The interactions between the nodes proceed along the links and each link has a trust value associated with it. That value is updated based on the interactions between the nodes at the two ends of the link. A node V can compute the trust value for another non-neighbor node W by aggregating trust values from other nodes in the following way:

1. enumerate (all) paths from W to V
2. aggregate trust values along the paths
3. merge the results of aggregation at V as the final trust value

The trust graph approaches vary in the details of the three above steps [60]: what domain is used to represent trust, what the selected paths are and what the aggregation and merging functions are. Trust values are either computed on demand between specific W and V or simultaneously for all nodes using some form of iterative methods that converge on an eigenvector of trust values.

2.4.2.2. Probabilistic estimation

The computations on trust graphs produce trust values that are hard to interpret. In particular, given a trust value for the node A it is hard to translate that value into the probability that A will cooperate. But this can be rectified if the assumption about probabilistic behavior of the nodes is made explicit and then well known probabilistic estimation techniques such as Bayesian estimation and maximum likelihood estimation are used to compute the trust of a given peer [62].

Another advantage of the probabilistic methods over the trust graphs is that the former bring a substantial reduction of the communication overhead. Most probabilistic approaches need to process only the feedback about the node of interest to determine the probability that it is cooperative, while the computations on the trust graph essentially aggregate all available feedback, i.e., opinions of all nodes about all other nodes.

In this thesis, we propose two robust routing protocols, one for the peer-to-peer overlays (§4) and another for the mobile ad-hoc networks (§5). Both of the protocols could be classified as a probabilistic estimation approach. In both of them, the nodes maintain reliability estimators for each of their neighbors in the network. These estimators are updated based on the evidence of cooperation or non-cooperation of the neighbors and the estimates are then used to make informed routing decisions to maximize the routing reliability.

2.4.2.3. Game-theoretic approach

In the game-theoretic approaches to reputation systems, it is assumed that the system consists of nodes that play a game in which the nodes make strategic decision on whether to cooperate or not and how to share and update the public reputation information. It is typically assumed the players are perfectly rational, i.e., that they are only interested in maximizing their own payoffs and indirectly there is a cost associate with having a low reputation, as the other players will not cooperate with a low reputation player. Knowing the payoffs and assuming rationality allows the nodes to compute Nash equilibria, which determine the behavior each of the nodes should take on to guarantee that no node will have any incentive to switch to a different behavior. Ideally, the Nash equilibrium should be a state in which all the nodes cooperate. The research in this area centers mostly on designing such mechanisms that guarantee the network's arrival in cooperative Nash equilibria.

There are many challenges that this basic game-theoretic approach faces. For the cooperation to happen, the nodes need to behave in the correct way as dictated by the Nash equilibrium. To compute the Nash equilibrium, the nodes must have enough information about the other nodes and the network they are in. It is very difficult to guarantee the completeness and accuracy of such information in the real-world distributed systems and large-scale networks. First, there is uncertainty about the value of cooperation to the other nodes [63], i.e., the payoff tables. Small changes in these values can dramatically sway the Nash equilibria. Second, players may not observe each other's actions but only their signals. In private monitoring games [64] the signals are different for different players, while in public monitoring games [65], all peers observe the same signals about the actions of the other peers. Third, the full rationality of the players cannot be expected. In particular, the behavior might be random. Finally, the Nash equilibria and, in effect, the behaviors may be prohibitively hard to compute (§ 2.4.2.5).

2.4.2.4. Evolutionary approach & reciprocity

Game theorists have also approached the problem of cooperation in a population of players interacting in a Prisonner's Dilemma (PD) setting from a more experimental angle. Most notably, Axelrod [66] has demonstrated the success of the tit-for-tat strategy in the evolutionary variant of PD. In this setting, pairs of players are involved in repeated PD games. Each player maintains a score, which is updated after every game round according to the PD payoff matrix. The players with the highest score are considered most fit and their strategies are replicated replacing other unfit strategies. The winning tit-for-tat strategy follows three simple rules:

1. *initially cooperate* - when interacting with an opponent for the first time, always cooperate

2. *punish* - if the opponent defected in the previous round, punish him by defecting
3. *forgive* - if the opponent cooperated in the previous round, cooperate even if there is a history of opponent's defection

The tit-for-tat strategy has been shown to be evolutionary stable, i.e., being able to drive into extinction small populations of invading defectors that try to exploit cooperators. At the same time, groups of tit-for-taters are always cooperating with each other, which allows them to accumulate score surplus, which, in turn, can be used to fight against transient groups of defectors.

To be successful, the tit-for-tat strategy needs a setting in which the PD interactions are repeated many times for the same pair of players, which allows punishment to occur. In a large population of individuals, repeated interactions might be infrequent. For example, on eBay, most of the transactions are one-off, it is rare that one buyer would purchase items from the same seller repeatedly. This observation led to the definition of a new setting in which every pair of players can only play one round of PD and never meet again. Building cooperation in this setting relies on the rule: "If A cooperates with B then B can reciprocate and cooperate with some other player C". This rule is termed *indirect reciprocity*, as opposed to the *direct reciprocity* rule followed by tit-for-tat.

In the case of indirect reciprocity, to build cooperation, players can no longer rely on private observation of the actions of the opponent. Once an observation is made, remembering that observation is pointless since all interactions are one-shot and such observation can never be used to make cooperation decisions. Hence there arises the need to exchange observations with other players, i.e., a reputation system. This can be implemented by associating a public label with each player. All players can read the label, and all players except the owner of the label are allowed to change it. It has been shown that to enable sustainable cooperation only two states of the label are sufficient [67]. The two states correspond to good and bad reputation. When a pair of players interacts, their labels are modified according to their actions. Ohstuka et. al. [68] systematize all the 4096 possible behaviors in a binary reputation system and study them experimentally. Out of all the behaviors, 8 of them lead to evolutionary stable cooperative strategies.

A population of agents using one of the “leading eight” strategies is able to sustain cooperation and drive out of existence any small population of defectors and/or reputation liars (i.e., players that set the labels to "bad" value even though their opponent cooperated).

There is a remarkable similarity between tit-for-tat and the leading eight strategies. The leading eight strategies exhibit all the properties of tit-for-tat: initial cooperation, forgiveness and punishment for defection. Tit-for-tat can be implemented with one bit of local state in the player, leading eight strategies make this state public by storing it in the player's label.

The fact that simple true/false reputation signaling is sufficient to support robust evolutionary stable cooperative systems has served as an inspiration in the design of the protocols in chapters 4 and 5. Both of the protocols use simple binary signaling to ensure routing robustness in ad-hoc networks.

2.4.2.5. Bounded rationality

Game theorists have considered imperfect monitoring games in which noise is allowed to occur in the system: imperfect observation of other players' actions, imperfect action execution, error-prone reputation information exchange, etc. Aside from these limitations, the players may have limited resources available to them to compute their behavior. A completely rational player is assumed to take on a behavior that follows one of the Nash equilibria in the system. However, Nash equilibria have been shown to be NP-hard to compute [69]. In the extreme, being unable to compute their behavior, the players can behave entirely irrationally or randomly from the point of view of other players.

The set of information exchange, perception, and computation limitations is commonly termed as *bounded rationality.* These limitations are faced not only by game-theoretic approaches to designing reputation-based systems for communication and computer networks, but in other settings as well. Conlisk [70] provides a plethora of empirical evidence from economics and experimental psychology in support of bounded rationality. The key observations are that:

- bounded rationality can explain a number of empirical anomalies in economics for which unbounded rationality models fail
- rationality is scarce, good decisions are costly, they require both reliable information, which is difficult to obtain and computational power
- bounded rationally leads people to imitate behaviors of others, which is cheaper than computing the behavior on their own

Why then, despite the bounded rationality and the unpredictable environment, do humans cooperate? Biologists and psychologists studying indirect reciprocity among humans have been trying to find the exact reasons for the remarkable stability of reputation and how it evolved [71, 72]. Many hypotheses have been proposed, most notably:

- group selection - Boyd et al. [73] suggest that cooperation can evolve by natural selection at the level of groups. Those groups that use reputation are more cooperative and hence more fit.
- conformist transmission - Heinrich et al. [74] show how weak conformity in populations can lead to the stabilization of reputation exchange and cooperation.
- costly signaling - Gintis et al. [75] show how using costly signals agents can advertise their quality as cooperators and in this way increase their reproductive success.

The fact that humans have evolved systems for stabilizing cooperation is key to our solution of the problem of freeriding in the peer-to-peer file-sharing networks that we propose in Chapter 7.

2.4.2.6. The importance of identity

Most reputation systems rely on the assumption that identities of the nodes are verifiable by all other nodes, however, in contrast to human societies, unless some technological constraints are put in place, the identities in open systems are low cost and easy to change. A malicious node whose reputation is low, can leave the system and rejoin under a different identity thus clearing the whole history of its defections. A malicious node can also assume a number of identities and have a significant presence in the network [48, 47]. Identity can also be stolen to take advantage of the reputation of the previous owner.

The solutions to this problem involve increasing the cost of cheap identities in some way. The reputation of the newly arriving peers can be initialized to a low value and making the peers have to gradually build up their reputation. The new peers can also be forced to solve a computational puzzle each time a new identity needs to be created, which would substantially increases the cost of a Sybil attack. These solutions are not particularly effective [48, 49]. To make the identities more persistent and verifiable, they can be managed in a centralized way, e.g., through the public key infrastructure based on a hierarchy of trusted Certificate Authorities as it is done in, for example, the Transport Layer Security (TLS) widely used for host authentication on the Internet [76]. However, in large-scale open networks, where every node has to be authenticated, and not only the servers, such solution might not be scalable or even feasible from the usability perspective. To address that problem, several decentralized alternatives have been considered.

In the *Web of Trust* approach, instead of a centralized certification authority the nodes certify each other's identities, i.e., one node cryptographically vouches for the identity of another. The certificates collectively form a graph. When a node i needs to verify the identity of some node j, then there must exist a path in the graph between i and j and to authenticate j, i verifies all the certificates along that path. A modern example of such a system is PGP [77], an email encryption and authentication application. In the Web of Trust approach, there is no centralized trusted authority issuing the certificates, any node can generate a certificate. Consequently, pathfinding in the certificate graph requires some additional reasoning about the trustworthiness of the certificates, not unlike in the case of the trust graph approach to reputation (§2.4.2.1). This increases the performance overheads and system complexity. Alternative distributed identity management systems have been proposed that do away with the graph and pathfinding. The certificates are stored and retrieved from random and independently chosen peers in the system [78]. Each certificate is replicated over many peers and the majority voting algorithm decides which is the correct certificate at query time.

In Chapter 7, similarly to PGP, we leverage the social identities and the social graph for trustworthy communication between the peers.

2.4.3. Market-based systems

Unlike the reputation-based approaches, the market-based systems do not rely on reciprocity, but on the payments that nodes make to other nodes for provided services. Payments are made in some form of currency. The system keeps track of the amount of currency that each node has, facilitates the currency transfers between the nodes and prevents double-spending, forgery or destruction of currency. Building such a system is particularly challenging in the context of open, ad-hoc networks such as peer-to-peer overlays or MANETs.

The most common class of solutions relies on a third trusted party for keeping track of the amount of currency and directly acting as a broker in transactions between the nodes. The simplest design with the most easily manageable security is the one where the brokering and banking functionality is centralized in one system component [79, 80, 81]. All the transactions have to pass through that component either synchronously during the transaction or asynchronously, by e.g., processing some form of a transaction receipt at a later time. In large-scale systems with high transaction rates the centralized component becomes the performance bottleneck; this is particularly true for the peer-to-peer systems.

The problems brought on by centralization have led to new approaches. To remove the bottleneck and the single point of failure, the banking functionality can be spread over several nodes [82, 83], each of which serves as a broker in the transactions. In other solutions, instead of relying on centrally issued currency, the nodes issue their own money or IOUs and keep track of them independently of other nodes [84, 85]. To avoid the bottlenecks introduced by the single point of trust, the trusted part of the system can also be distributed by equipping the nodes with trusted tamper-proof modules that keep track of the available credit at each node [86].

The reputation-based and market-based approaches are remarkably similar in their design. Both of these systems store some state for each node: a reputation value, a summary of the node's behavior, the current balance on the currency account or a set of signed IOUs from other nodes. Some authors have pointed out that reputation is, in fact, a form of currency [87]. Whatever the rules governing the changes in the reputation or currency values, they have to be stored and updated in a secure fashion. While it is fairly straightforward to delegate that to a centralized trusted system component, it does not scale well with the system size and introduces a single point of failure. On the other hand decentralizing the reputation and currency state management creates new system availability and security challenges, which so far have not seen a silver bullet solution.

In Chapter 7, we show how instead of constructing a technological solution to the problem of freeriding in a peer-to-peer file-sharing system, the already existing human reputation system can be leveraged to ensure cooperation of the nodes in a computer network.

Chapter 3

Generic Emergent Overlays in Arbitrary Metric Spaces

3.1. Introduction

The important first step in designing networks for efficient and robust information flows is to define a network topology that can support them along with the appropriate mechanisms for formation and maintenance of that topology. In this chapter, we address this problem in the context of the peer-to-peer overlay networks (§2.2), ad-hoc networks of computers communicating over the Internet.

Most of the state-of-the-art structured overlay networks [36, 35, 88, 89, 37, 90, 39, 91] follow a similar design paradigm. First, a global network structure is defined and the peers are placed in some identifier space. The structure has properties desirable from the application's point of view such as high routing path redundancy, increasing the resilience to failures and routing path lengths that scale logarithmically in terms of the network size. Then, this global network structure is expressed as invariants, which, as nodes join and leave, are maintained by each node locally to ensure the coherence of the global network topology. Structured overlays, despite their performance guarantees, are driven by complex distributed protocols and the applications have little flexibility and control over how the overlay topology is formed.

In contrast, in unstructured overlays [34, 33], the design process starts from the local goals and rules without any particular target global structure in mind. The resulting algorithms and protocols are simple and offer much flexibility in forming the overlay topology and routing the messages, however they lack the routing efficiency and scaling guarantees of structured overlays.

In this chapter, we take an approach to overlay network maintenance that combines the flexibility and ease of implementation of the unstructured overlays with the efficiency and scalability of the structured overlays.

As we have observed in §2.2.2.2, the common characteristic of all structured overlays is that each node is endowed with an identifier and there is some notion of distance defined between the identifiers. Through the distance function, each node can know its position relative to other nodes in the topology. In each routing hop, the message routed in the overlay is brought closer to its destination in terms of the identifier space distance. To ensure the progress of messages towards the destinations each node maintains a set of connections. The global overlay connection topology guarantees efficient and scalable delivery of messages.

Another observation is that the social, transportation, biological and other complex networks rely on some form of greedy routing for propagating information from one node to another (§2.1.3). Similarly to peer-to-peer overlays, the nodes are embedded in a hidden metric space of node attributes [1] and the notion of distance in that space is used for greedy routing. This leads to a question: given the success of greedy routing and its flexibility to work in a large diversity of metric spaces, can we engineer a similar routing solution for the peer-to-peer overlays?

To address this problem, we propose a generic overlay routing and maintenance algorithm in which the application can define its own overlay identifier distance function. Our algorithm relies solely on the knowledge of the identifier distance function and does not specify any global topology that needs to be maintained. The global topology emerges in a self-organized way as a result of a simple connection opening rule.

The node identifiers are selected uniformly at random from the identifier space. In each routing step we greedily route the message to the next hop that is closest to the destination. We compute the rate at which the message approaches its destination, i.e., how much the distance to the destination is shortened during one hop. We require that the rate for each hop be at least γ, a design parameter. If this condition is not satisfied then routing is inefficient and additional connections are opened by the maintenance algorithm to ensure the minimal rate of message progression towards the destinations.

To verify the claim that the above simple algorithm is indeed able to form an efficient overlay, we evaluate our overlay by simulation. We observe that:

- The resulting overlay has logarithmic scaling properties. Both the average routing path length and the average node degree are logarithmic in terms of the number of nodes in the network.
- The routing path length vs. node degree tradeoff can be controlled by adjusting the single design parameter γ.
- Logarithmic overlay scaling can be achieved for any identifier distance metric by adjusting γ.
- Local structures emerge tightly interconnecting nodes in the identifier space on short distances. This common characteristic is shared by all state-of-the-art structured overlays and is crucial for e.g., last hop routing and key replica management in DHTs.
- The overlay is robust and has low maintenance overhead even in presence of high churn.

The overlay networks generated by our algorithm have characteristics comparable to many of the well-known approaches, while offering a number of additional advantages:

- Up to our knowledge it is the first overlay maintenance algorithm that is driven by the overlay traffic, i.e. connections are created only when they are needed.
- The abstract space of node identifiers and distances between them generalizes over the previous approaches.
- In contrast to other structured approaches there is no pre-defined rigid global structure that has to be maintained. In our case, the topology emerges in a self-organized way as a function of the underlying identifier space. The lack of pre-defined topology greatly simplifies the algorithm and minimizes the implementation effort.
- The proposed algorithm leaves plenty of room for adjusting the routing efficiency and the number of connections the nodes need to maintain, this combined with the generalized identifier space gives a level of customizability not available in other overlays.

3.2. The model

In this section we present the basic assumptions followed by the formulation of the proposed overlay routing and maintenance algorithm.

3.2.1. Graph embedded in a metric space

Let the network be represented by a graph $G(V, E)$, where V is the set of overlay nodes and E is the set of overlay connections between them. Let $id : V \to I$ be a function that assigns an identifier from the set I to each of the nodes in V. Let $d : I \times I \to \mathbb{R}$ be the distance function. The id function embeds the nodes in the metric space defined by d, hence d must satisfy the four properties of the distance function in a metric space: non-negativity, symmetry, identity and the triangle inequality. The pair (I, d) is the *identifier space* and the *mapping function* id maps the overlay nodes into that space.

The communication in the network proceeds by sending messages. The messages can only be sent along connections. Once a connection is established between two nodes it can be used to send messages in both directions.

3.2.2. Routing

Assume a source node $m.src$ wants to send a message m to a destination node $m.dest$ ($m.f$ denotes the field f in message m). The routing proceeds in the standard hop-by-hop way. The next hop is selected by greedily minimizing the identifier space distance to $m.dest$ and at the same time avoiding previously visited nodes.

Let $m.visited$ be the set of nodes through which m has already been routed. Let v_c be the node that currently holds m and needs to forward it. Let $neigh(v_c)$ be the set of neighbors of v_c. The node v_c selects the next hop v_{nh} based on the following rule: take the set $neigh(v_c) \setminus m.visited$, and from it select node v_{nh} for which the value $d(id(v_{nh}), id(v_d))$ is the lowest. After selecting the next hop, v_c adds v_{nh} to $m.visited$ and forwards m to v_{nh}.

The following special cases occur:

- **NHimp** - next hop impossible - a message reaches a dead end, $neigh(v_c) \setminus m.visited$ is an empty set, the message is dropped

- **TTL0** - TTL zero - each message has a time-to-live counter decremented with every hop, when it reaches zero the message is dropped

3.2.3. Overlay maintenance

The maintenance in our overlay is driven by routing. To ensure eventual delivery, each routing hop should advance messages closer towards the destination in the space defined by d. What is more, this advancement should occur at a certain minimal rate of progression to provide efficient overlay message delivery, otherwise new connections have to be created to ensure that this happens. We base our overlay maintenance algorithm on this simple maintenance rule.

For a given next hop v_{nh} from the current node v_c towards the destination $m.dest$ we define the *routing convergence rate* as:

$$cvg(v_c, v_{nh}, m.dest) = \frac{d(id(v_c), id(p.dest))}{d(id(v_{nh}), id(m.dest))} \tag{3.1}$$

Let γ be the minimum required routing convergence rate, a design parameter. Routing convergence rate is the measure of how much the next hop shortens the distance to the destination. If a hop $(v_c \rightarrow v_{nh}, m.dest)$ does not satisfy the condition $cvg(v_c, v_{nh}, m.dest) \geq \gamma$ then that hop is *weak*, otherwise it is *strong*.

When a weak hop is encountered while routing a message m, the maintenance protocol sends a connection request $cr = (v_c, p.dest)$ with the destination set to $m.dest$ indicating the origin of the request as v_c, the current node. The connection request is routed towards the destination normally as other messages in the greedy self-avoiding way. When some node v_{resp} receives a connection request $cr = (v_o, dest)$ and if the hop $(v_o \rightarrow v_{resp}, dest)$ is not weak then v_{resp} responds to v_o with connection acknowledgement and the connection between v_o and v_{resp} is established and the routing of cr stops. Otherwise if $(v_o \rightarrow v_{resp}, dest)$ is weak, the cr continues to be routed.

When a timeout happens while sending on one of the connections then the sender closes that connection and removes the recipient from its neighbor set.

3.2.4. Maintenance suppression

Every node keeps track of the connection requests that it has sent until either the corresponding connection response arrives or a timeout occurs. This request-response tracking has the following purpose. Consider the time between two events: (1) the sending of a connection request $c(v_c, dest)$ by node v_c and (2) the receipt of the corresponding connection response. Assume additionally that there are no connection requests being sent or responses arriving during that time. However, there may be many messages with the same destination $dest$ arriving at v_c. According to the proposed maintenance algorithm each of these messages takes a weak next hop and triggers a connection request, which is identical to the one already sent earlier. This would lead to the generation of many unnecessary connection requests.

To prevent this from happening, whenever some $cr = (v_c, dest)$ is about to be sent the list RL of requests currently awaiting their responses is checked. If any of the potential responders to the connection requests on RL is also a valid responder to cr, then cr is not sent. Let $cr' = (v_c, dest') \in RL$ be some connection request awaiting its response. Let v_r be the potential responder to cr', v_r must satisfy the condition:

$$cvg(v_c, v_r, dest') \geq \gamma \tag{3.2}$$

If v_r is also a valid responder for cr, then it also satisfies:

$$cvg(v_c, v_r, dest) \geq \gamma \tag{3.3}$$

We also know that the three identifiers of the three nodes $(v_r, dest, dest')$ must satisfy the triangle inequality:

$$d(id(v_r), id(dest)) + d(id(v_r), id(dest')) > d(id(dest), id(dest')) \tag{3.4}$$

Combining 3.2, 3.3 and 3.4 we obtain:

$$\gamma d(id(dest), id(dest')) < d(id(v_c), id(dest)) + d(id(v_c), id(dest')) \tag{3.5}$$

If there is any $cr' = (v_c, dest') \in RL$ for which 3.5 is true then $cr = (v_c, dest)$ is not sent.

The operation of our overlay routing and maintenance algorithm has been summarized in Algorithm 1. Note that for clarity the handling of timeouts, TTL0 and NHimp events has been omitted.

3.3. Simulation results

In this section we examine the performance of our routing and overlay maintenance algorithms experimentally.

Algorithm 1: Overlay routing and maintenance algorithm for arbitrary peer identifier spaces

```
initialize
    $RL \leftarrow \emptyset$ ;
    $neigh \leftarrow$ initialize with peers via bootstrap mechanism;
// application calls this function to send messages
function sendMessage(payload,dest)
    forwardMessage(Message(payload,self,dest,$\emptyset$));

function forwardMessage(Message(payload,src,dest,visited))
    $next\_hop = argmin_{x \in neigh \setminus visited} d(id(x), id(dest))$;
    send $Message(payload,src,dest,visited \cup self)$ to $next\_hop$ ;
    if $payload$ is $ConnectionRequest$ then
        return ;
    end
    // maintenance is triggered by weak hops
    // only for payload that is not a ConnectionRequest
    if $\frac{d(id(self),id(dest))}{d(id(next\_hop),id(dest))} < \gamma$ then
        // check the maintenance suppression condition
        if $\neg\exists_{cr'(self,dest') \in RL} \gamma d(id(dest), id(dest')) <$
        $d(id(self), id(dest)) + d(id(self), id(dest'))$ then
            $RL \leftarrow RL \cup cr(self, dest)$;
            // route the connection request as a new message
            forwardMessage(Message(ConnectionRequest(self,dest),self,dest,$\emptyset$));
        end
    end
receive $Message(payload,src,dest,visited)$
    if $payload$ is $ConnectionRequest(origin,dest)$ then
        // check if can accept the connection request
        if $\frac{d(id(origin),id(dest))}{d(id(self),id(dest))} \geq \gamma$ then
            send $ConnectionResponse(self,dest)$ to $origin$ ;
            return ;
        end
    else if $dest{=}self$ then
        deliver Message to application;
        return ;
    end
    forwardMessage(Message(payload,src,dest,visited));
receive $ConnectionResponse(responder,dest)$
    $RL \leftarrow RL \setminus cr(self, dest)$;
    $neigh \leftarrow neigh \cup responder$;
```

3.3.1. Experimental setup

We use ProtoPeer (§A), an event-driven simulator. Each node in the simulated network generates messages in a Poisson process. The generation rates are identical across all the nodes. A node v_i generates a message m with the destination $v_j \neq v_i$ selected uniformly randomly.

The average message generation rate at all nodes is set to one message every second. All messages, connection requests and connection responses are delivered with a latency uniformly distributed between 100ms and 200ms. The time-to-live for overlay messages is set to 100 hops. We do not explicitly simulate the underlying network topology. This simple network model is sufficient for verifying the correctness of our algorithm and the structural properties of the overlay, performance testing in a more realistic setting is left as future work (§3.4).

The bootstrap process is as follows. Each simulation begins with a set of 30 nodes interconnected uniformly randomly with an average degree of 5. The size of this initial network is large enough to ensure that it remains a connected graph even if some of the initial nodes depart at the beginning of the simulation. Each new joining node connects to 5 uniformly randomly selected neighbors from the network. These bootstrap connection requests and responses are delivered directly and are not routed in the overlay. In a concrete implementation the peers would be bootstrapping from a known host list (e.g., downloaded from the Web), we do not simulate this process in detail since our overlay is not sensitive to the choice of initial neighbors for the peer.

To demonstrate that the results are independent of the chosen identifier space we select five representative spaces for the experiments:

- 1D - one dimensional ring, as in Chord [36], $I = [0, 1)$, $d(a, b) = min_{k \in \{-1,0,1\}} |a - b + k|$
- 2D - two dimensional spherical coordinates, the identifiers are placed on the sphere $I = [0, 2\pi)^2$ and the shortest distance is measured along the great circle crossing the two identifiers
- 3D - three dimensional Euclidean space with wraparound (surface of a 4D hypertorus).
- PFX - prefix routing as in Pastry[89] with the identifier space of 128bit vectors, assume we are computing $d(a, b)$, bits in a and b are compared from the highest order bit to the lowest order bit, if i is the index of the first bit which differs between a and b, then $d(a, b) = 2^i$
- XOR - XOR distance metric as in Kademlia [37], an XOR of two identifiers is computed and the result is taken as an integer distance value with 160 bits

For all the identifier spaces the nodes select their identifier uniformly randomly out of the set of all possible identifiers.

The arrivals and departures of the nodes (churn) are simulated. Arrivals are a Poisson process with a default average rate of 0.002 nodes per second. The lifetime of the nodes is power-law distributed [92] with the minimal lifetime of 10s and the exponent of -1.2.

During the simulation we track the number of TTL0 and NHimp events (§3.2.2). The churn is low in most simulations. We devote Section 3.3.3 to the study of routing failures under high churn conditions.

3.3.2. Scaling

To test the fundamental scaling properties of the overlay we let it increase in size over time by setting the minimal node lifetime to a higher value of 50s. For different network sizes and identifier spaces we measure the average routing path length and the average and maximum degrees. The average path length is measured over all the messages that have reached their destinations. This excludes the connection requests and responses. The node degrees are measured on the snapshot of the overlay topology at the end of a measurement epoch.

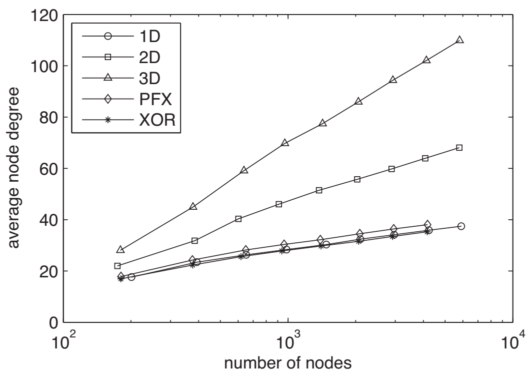
The measurements are plotted on Fig. 3.1. Both the average path length and the average degree scale logarithmically in terms of the number of nodes as in the state-of-the-art overlays. In addition the logarithmic scaling of the maximum degree is evidence for a balanced degree distribution, which is crucial for balancing the message forwarding load among the peers.

The case of the 3D identifier space is an outlier in our scaling experiments. In contrast to other identifier spaces, the average degree is rising much more rapidly. This is caused by the "curse of dimensionality" problem, which we discuss in §3.3.5.

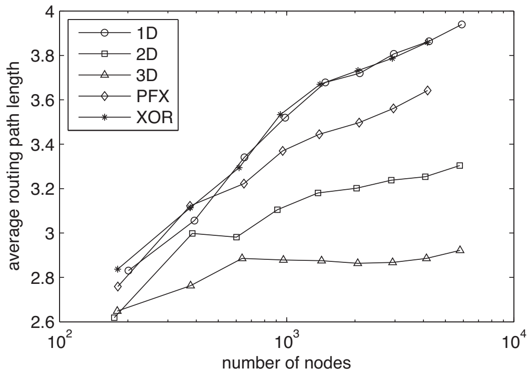
3.3.3. Maintenance overhead and failures in extreme churn conditions

Apart from scaling, another important characteristic of an overlay is its maintenance overhead and resilience to node departures or failures. To measure these characteristics we switch from the power-law node lifetime model to Poisson arrivals and departures, the rate of arrivals and the rate of departures are gradually increased. While this happens the network size is kept around 1000 nodes. To stress test the overlay we set the churn rates to values that are considerably higher than those typically seen in peer-to-peer network deployments.

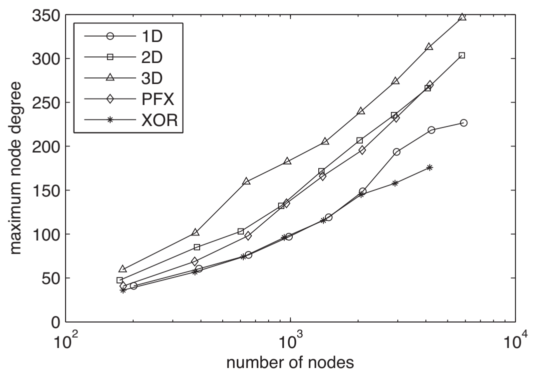
Figure 3.2 summarizes the results. Our overlay maintenance algorithm keeps the routing failures under 0.2% even when 40% of the nodes are replaced with new ones every minute. For small churn rates the maintenance traffic increases faster with the increasing churn, when the churn is higher the massive parallelism in connection request sending lowers the number of connection requests a newly joined node needs to open as it is more likely to receive connection requests from other newly joined nodes. As the churn rate increases the average degree decreases, there are more missing connections in the topology caused by churn. Even though some poorly connected nodes might lay on the routing path, its average length increases only slightly in high churn conditions.



(a) Average node degree



(b) Average routing path length



(c) Maximum node degree

Figure 3.1. The scaling of the average path length and the node degree. Each data point is an average of 20 measurements. Standard is under 10%, omitted for clarity.

The overall robustness of our overlay is high, the overlay does not lose connectivity, high routing efficiency and low failure rates are maintained.

3.3.4. Varying the γ parameter

The routing convergence parameter γ is crucial in our algorithm. We explore experimentally how the changes of this parameter influence the performance of the overlay. For γ values varying from 0 to 4 and for the different identifier spaces we grow the network until it reaches 1000 nodes. For the resulting network we measure the average number of hops and the average node degree.

The results show (Fig. 3.3) that adjusting the γ parameter allows for precise control of the path length vs. degree tradeoff. Distinct operational regimes can be defined:

- **low degree** - $\gamma \ll 1.0$ - most of the hops are strong and only a few new connections are opened, message routing relies more on the self-avoidance property of the routing

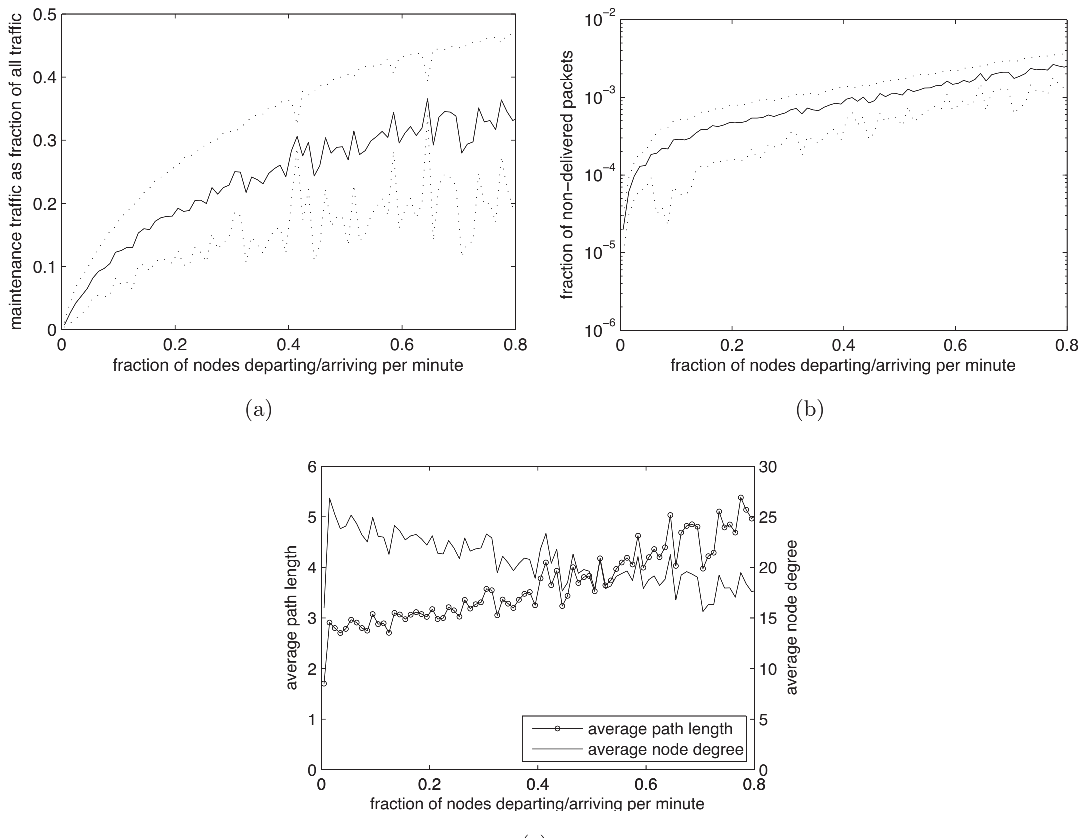


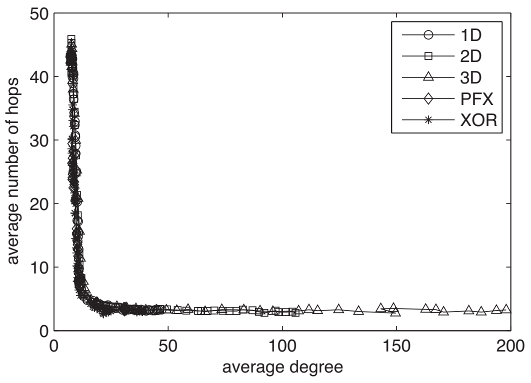
Figure 3.2. Maintenance cost and failure rates under extreme churn conditions. Results from 20 independent experiments. Standard deviations marked with the dotted lines.

algorithm, many nodes need to be visited as the routing gradually and mostly randomly converges towards the destination, messages are frequently dropped due to the TTL0 and NHimp events (§3.2.2)

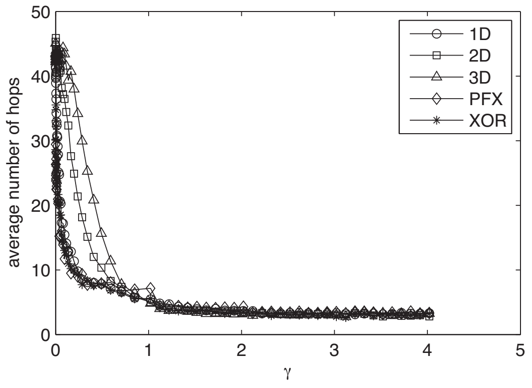
- **high degree** - $\gamma \gg 1.0$ - most of the hops are weak and a large number of connections needs to be opened to form strong hops to the different areas of the identifier space, the convergence of a message is guaranteed by the high γ value, the distance to the destination exponentially decreases (at least by the γ factor in each hop), self-avoidance rarely has to be used and the average path length is small

- **balanced** - $\gamma \approx 1.0$ - in this regime the scaling of both the average degree and the average number of hops are logarithmic in terms of the number of nodes as demonstrated in §3.3.2.

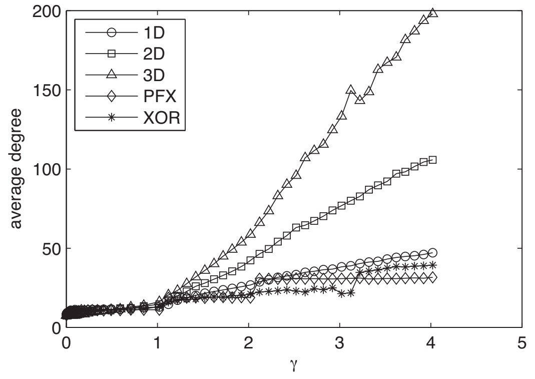
In the next section we discuss the high degree regime further and provide a way for selecting the γ parameter such that the overlay operates in the balanced regime.



(a)



(b)



(c)

Figure 3.3. The influence of varying the γ parameter on the performance of the overlay. Each point represents a measurement on a separate overlay that was independently run.

3.3.5. Routability

For a given identifier space, only for some values of γ, the overlay is in the balanced regime and we clearly need a way of determining these values. To achieve this we define the concept of routability. The *routability* $R(\gamma, I, d)$ of the identifier space (I, d) under the convergence rate γ is the expected probability of finding a strong next hop taken over all possible sources and destinations. Let X be the random variable describing the distance between two random identifiers in the (I, d) space and let $p(x)$ be the probability density function of X, then $R(\gamma, I, d)$ is:

$$\int_0^{x_{max}} p(x)\mathbb{P}(0 < X < \frac{x}{\gamma})dx \tag{3.6}$$

The values of R are computed numerically for the different spaces and values of γ (Fig. 3.4). Low routability values indicate that the network will operate in the high degree regime and vice versa high routability is a good predictor of the low degree regime (compare these results

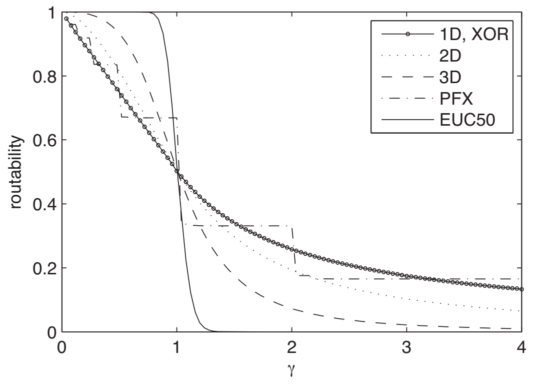


Figure 3.4. Routability for the different identifier spaces (defined in §3.3.1) and values of γ.

to Fig. 3.3).

To provide an extreme case we have considered a highly dimensional space EUC50, the surface of a 51-dimensional hypertorus with Euclidean distance metric. The value of routability for EUC50 at $\gamma = 2.0$ is very close to 0.0 and for $\gamma = 1.1$ it is 0.17 which we have verified experimentally to be enough to provide logarithmic scaling. This result also demonstrates that highly dimensional spaces are not good a good choice for routing due to their "curse of dimensionality"[1]. Only when the node degree is very high can efficient routing be achieved.

The routability concept can be conveniently used to find the ranges of γ values and identifier spaces for which the network operates in the balanced regime, i.e., with routability values in the mid-range, close to 0.5. It has to be noted that this is a necessary condition for good scaling of the network, not a sufficient one.

Routability depends on the variable X which among other factors depends on the distribution of the identifiers in the identifier space, which thus far was assumed to be uniform. If that distribution is skewed, this may greatly influence the routability value. The routability formula should also factor in the actual message traffic distribution which we assumed to be uniformly distributed over the set of all source-destination pairs. Exploring these dependencies is beyond the scope of the thesis and is left as future work.

3.3.6. The emergence of local structures

Some overlays maintain a completely deterministic set of neighbors, others create random links [91]. However, all of the current structured approaches maintain at least one deterministic connection, usually to its closest neighbor. Those connections are eagerly maintained and are crucial to reliable routing, especially at the very last hop. In our routing and maintenance algorithm, the creation of this type of connections is not explicitly a part of the algorithm. However, nodes following the simple local rule for sending the connection requests create a

[1] `http://en.wikipedia.org/wiki/Curse_of_dimensionality`

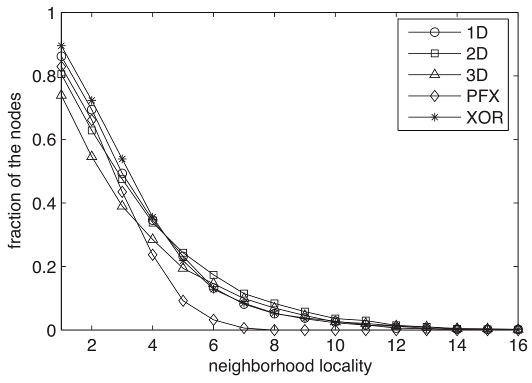


Figure 3.5. The fraction of nodes with a given locality.

dense network of short-range links.

To measure this effect, we define the value of *locality* for each node. With a vertex $v_i \in V$ we associate the series of vertices $l_{i1}, l_{i2}, ...l_{i(n-1)}$ where $n = |V|$, and:

$$\forall_{i,j,k} d(id(v_i), id(l_{ij})) < d(id(v_i), id(l_{ik})) \Rightarrow j < k \qquad (3.7)$$

Then locality of v_i is equal to m if $\forall_{k=1..m} l_{ik} \in neigh(v_i)$. Informally, the locality of v_i is the number of nodes closest to v_i in the identifier space such that all of these closest nodes are connected to v_i.

We take the topology snapshots of the overlays constructed in different identifier spaces with 1000 nodes. We measure the fraction of nodes with the given value of locality (Fig. 3.5). The topology snapshot is taken while the overlay is churning and thus some of the nodes may have just joined the overlay and have not opened a sufficient number of connections or some of the nodes may have lost a connection due to a neighbor's departure and have not replaced this missing connection with another one. Churn decreases the measured fractions of nodes. Despite churn, 80-90% of nodes are connected to their closest neighbor, and 70-80% of the nodes are connected to two of their closest neighbors. In a 1D ring identifier space if all of the nodes have locality 2 then there exists a global ring spanning all of the nodes, it is possible to visit all the nodes by hopping along that ring in one of the two directions. In overlays based on the 1D ring identifier space (e.g., Chord [36]) this global spanning ring is explicitly maintained. In the case of our algorithm, the ring emerges as a result of the simple connection opening rule in the maintenance algorithm. This rule is independent of the identifier space used and similar local structures are universally created in identifier spaces other than the one-dimensional ring.

3.4. Discussion of results and future work

3.4.1. Topological flexibility through γ adjustments

We have shown how to use the concept of routability to find the range of γ parameters that ensure that the network stays in the balanced operating regime for a given identifier space. We verified experimentally that this regime exhibits optimal overlay scaling properties, however one might ask a question why do the values of γ close to 1.0 still produce networks with logarithmically increasing routing path length, despite the fact that each hop is not required to shorten the distance to the destination. This is explained by the fact that each node has a non-negligible probability of opening a long range link which acts as an effective shortcut in the identifier space. Though there are only a few of those long range links in the case when γ is close to 1.0 they significantly lower the average path length.

Throughout the chapter we have assumed that γ is a system-wide constant. However, it may vary in the following ways:

- **per node** - each node might locally decide at what routing convergence rate it forwards the messages, indirectly this gives the node a way of controlling its degree
- **over time** - γ can change to adapt to changing network conditions, e.g., churn
- **per message** - some types of traffic may be prioritized, e.g., traffic with a specific destination might have a higher routing convergence rate associated with it

Fine-grained control over γ gives a considerable degree of flexibility in shaping the overlay, adjusting the length of the routing paths for different types of traffic and deciding which nodes the traffic traverses through. This may be particularly useful in cases when the source-destination distribution of the message traffic is skewed.

3.4.2. Optimizations

In our maintenance algorithm the connection requests are triggered by messages and are normally sent immediately after the message itself towards the same destination as the message. It is very likely that both the connection request and the message that triggered it follow the same routing path. This can be exploited to piggyback the connection requests on the actual application messages. This can be easily implemented since each message already contains the visited list, the connection request can then be a single bit flag attached to a node in the visited list. We plan to implement connection request piggybacking and investigate how much maintenance bandwidth can be saved in this way.

On the other hand, our protocol adds additional overhead to each of the messages by storing the visited list. The size of this list equals to the number of hops and scales logarithmically in network size. The list is used to prevent self looping while routing. However, once the overlay is stable there are no loops and the visited lists are not used. For $\gamma > 1$ Each routing loop has at least one weak hop, this suggests that it may be sufficient to store visited nodes only when the next hop is weak. Moreover, instead of explicitly storing the whole list of visited nodes, Bloom filters on node identifiers could be used to drastically decrease the space needed for storing the list. These optimizations await experimental investigation.

If our overlay is used as the routing substrate for the distributed hash tables, the emergent local structures (§3.3.6) can be used to manage the replicas of the key-value pairs. Furthermore, the node identifier space can be selected such that it better suits the needs of a particular DHT key space and the application that uses the DHT. We plan to implement a DHT on top of our overlay and investigate the advantages brought by flexible identifier spaces.

3.5. Related work

Our approach is most similar to Freenet [34], which follows simple rules for opening new connections. Moreover, in Freenet, just as in our approach, routing is loop avoiding. However, in Freenet loop avoidance is implemented by keeping the routing state at the nodes, while in our algorithm we keep it in the message, which is a stronger form of loop avoidance. Another major difference is that in Freenet new connections are opened by performing a random walk when the node joins a network. This leads to a scale-free topology with a power-law node degree distribution with a small number of nodes having a very high degree. In our case the maximum degree increases only logarithmically with the overlay size.

We have shown how adjusting γ can be used to trade off between the node degree and the routing path length. In a real network implementation this corresponds to the maintenance bandwidth vs. routing latency tradeoff. This problem has been studied in the context of Accordion [93], where given a bandwidth budget the protocol adjust the number of maintained connections to the current churn levels. The obtained tradeoff curves are similar to the ones in Fig. 3.3. A simplified version of this algorithm can be implemented in our overlay by making γ a function of the current churn level and the given bandwidth budget.

A widely studied topic in the domain of complex networks are the small-world graphs. They are commonly defined [94, 95] as graphs having both a small diameter and high clustering coefficient. Our overlay satisfies both of these properties, except to quantify the clustering we employed our own measure - the node locality (§3.3.6). Our overlay can be viewed as a dynamic model for small-world growth. The first dynamic model capable of generating networks having small-world properties was proposed by Watts and Strogatz [13]. In their approach they start with a regular graph and then modify its structure through random rewiring. The probability of rewiring can be controlled. As the probability increases, the network goes through three topologically distinct stages. First, the topology is highly regular with high average path length, then it is small world and finally completely random with short path lengths but low clustering coefficient. Kleinberg [28] places the nodes on regular multi-dimensional lattices and addresses the problem of connection length distribution that would ensure efficient routing. We generalize on this work further by considering a wider family of spaces in which nodes are embedded and proposing a concrete routing and network formation algorithm that is able to achieve logarithmic routing path scaling.

3.6. Conclusions

In Gnutella [33], requests are flooded through the network until they reach their destination. The nodes are not embedded in any space. What gives the structured overlays their structure is the space the nodes are embedded in. Once the space is added, the flow of messages is given directionality and they no longer have to move in all directions simultaneously to find their destinations but only towards directions that decrease their distance to the destinations. A form of this greedy routing is used in all state-of-the-art structured overlays and in most of the real-world complex networks. Routing decisions are made solely based on local measurements of gradients in the identifier space. The knowledge of the properties of the whole space is not necessary. We have proposed a generic algorithm that for any metric identifier space is able to maintain the overlay and route messages based only on local information and simple rules.

Each overlay runs a maintenance algorithm that opens connections to ensure that for every node every forwarded message is brought closer to the destination in terms of the space distance. Our overlay maintenance algorithm generalizes this rule. The proposed algorithm is parametrized by γ which allows for precise control of the path length vs. degree tradeoff while generating relatively small maintenance traffic even under high churn.

Normally, the overlay maintenance algorithm keeps the network prepared for efficient traffic routing from any source to any destination. In our algorithm, maintenance is tied to routing and connections are created on demand and only cover the set of destinations that actually appear in the traffic without opening unnecessary connections. This may be controlled at a finer granularity by associating different γ values with different forwarding nodes and different types of routed traffic.

The main insight of our work is that the identifier space is flexible and the properties of the individual identifier spaces used in structured overlays are not significant as long as the spaces provide a consistent local gradient for the greedy routing algorithm to follow. Moreover, the topology of the overlay does not have to be an eagerly maintained pre-defined rigid structure but can emerge in a self-organized way from simple local maintenance rules that adapt the topology to the identifier space. This departure from the structural rigidity of the overlays opens new possibilities of handling skewed overlay traffic distributions, prioritizing traffic, adapting to inhomogeneous allocation of node identifiers and customizing the identifier space to the needs of the application.

Publications: [96]

Chapter 4

Bandwidth-efficient Fault-tolerant Routing in Peer-to-peer Overlays

4.1. Introduction

In the previous chapter, we have defined a generic peer-to-peer overlay formation and maintenance algorithm that constructs efficiently routing topologies. The overlay tolerates peer departures and persistent link failures by opening additional connections to repair the topology. Similar mechanisms for tolerating persistent failures exist in other P2P overlays (§2.2.3). However, any sufficiently large P2P system will also experience non-persistent, intermittent message loss due to various factors, including peer overload [97], transient network connectivity problems [98] or DDoS attacks [99]. These problems cannot be solved using the same techniques as for the persistent failures, since dealing with this type of failures incurs substantial additional repair costs such as the overlay search for new neighbors or additional key replication in distributed hash tables [100].

Transient failures pose a challenge. When message loss occurs at some peers or network links, even with low probability, it may substantially decrease the overlay routing reliability, especially for longer routing paths. This can be demonstrated using a simple Bernoulli trial argument as in [99, 46]. Moreover, some peers might introduce message forwarding delays. The delays accumulate over the whole path and the delivery time may exceed application timeouts.

A simple local solution to the problem could use hop-by-hop acknowledgments. After some peer receives the message, it sends an acknowledgement back to the previous hop indicating that it has processed the message successfully. By keeping track of the acknowledgements the peer can eliminate the faulty peers from the routing paths. However, a receipt of an acknowledgement does not guarantee that its sender has already received the acknowledgment from the next hop on the path. The acknowledgment thus signals only that the message performed one hop and not that it successfully traversed the whole path and reached the destination. All the nodes on the path must be functioning correctly to complete the message delivery. The loss probability and delays are cumulative over the whole routing path and such failures are challenging to detect by relying only on local acknowledgment signaling. Path-wide mechanisms are necessary to achieve this.

In practice, all the widely used solutions to the problem of reliable message delivery rely on redundancy for masking the message loss and delays. In one approach [46], messages are sent along several disjoint routing paths. If one path fails, the messages on the other paths can still potentially reach the destination. Another approach [37] uses iterative routing that keeps several parallel RPCs (remote procedure calls), each independently asking peers on the routing path for the next hops that are closer to the destination. When one RPC fails the other can still make progress. These solutions heavily rely on message redundancy and thus have a high bandwidth cost. Moreover, they do not in any way remember where the failures happen and are likely to repeat the same routing mistakes again.

In this chapter we propose an overlay routing protocol that routes each message only along a single routing path for bandwidth-efficiency, detects message loss along the paths and over time learns to avoid the lossy and slow peers in an entirely decentralized and self-organized way.

In our Forward Feedback Protocol (FFP) each routed message is followed on its routing path by a binary feedback message (§4.3.2). The feedback is positive when the message is delivered on time and negative when routing path delays exceed the application timeout or when messages are lost. Based on the feedback, each node on the path locally and independently learns to avoid slow or lossy parts of the overlay. Despite the independence of its nodes, the overlay as a whole converges on reliable loop-free routes. The process of learning from feedback is continuous. The system constantly self-organizes in response to changing delay and loss conditions.

The proposed FFP protocol combines several properties in a novel way:

- **path-wide fault-tolerance** - FFP's failure detection covers the whole routing process: from the moment the source sends the message, until the final destination acknowledges the receipt. Thanks to that FFP can respond to failures that can only be detected at the routing path level, such as when the delay accumulated over the whole path exceeds the application timeouts.

- **topology-oblivious routing** - FFP peers do not require any prior knowledge about their neighbors, including their location in the overlay address space (§4.3.3). Routing is topology-oblivious. Through feedback, each peer individually learns which of its neighbors is a reliable forwarder for which destination. The network as a whole converges on efficient routing paths (§4.5.3) in a decentralized self-organized way.

- **low overhead** - peers in our system continuously learn through feedback and remember past failures. This is in contrast to some of the existing approaches which rely on multiple redundant paths for increasing fault-tolerance and do not learn from routing mistakes. We demonstrate in live Internet deployments (§4.5) that despite the 2-5 times lower bandwidth usage, our solution achieves the success rate comparable to the existing redundancy-based approaches.

- **scalability** - FFP is fully decentralized and scalable, we show the local state size to be only $O(\log^2(N))$ in terms of the network size (§4.4).
- **universality** - FFP is simple and general enough to be used as a fault-tolerance mechanism in any recursively routing overlay or in any recursively routing network (§4.6).

The remainder of the chapter is organized as follows. In §4.2 we go over the existing work. Section 4.3 describes the basic system setup followed by the specification of the Forward Feedback Protocol (§4.3.2). We then analyze the protocol's scalability and propose several optimizations (§4.4). FFP is then evaluated (§4.5) together with the existing approaches under a range of failure scenarios and workloads and the chapter concludes in §4.7.

4.2. Related work

Considerable attention has been devoted to overlay fault tolerance to churn, i.e., constant peer arrivals and departures. Many designs have been proposed [36, 39, 35, 88, 89, 100] each with different topologies and overlay maintenance algorithms (§2.2). The problem of message loss is solved either by retries or by falling back on multipath or iterative routing, which we explain next.

Although our solution is based on recursive routing, many of the widely deployed systems such as Coral [101] or Kademlia [37] rely solely on iterative routing. *Iterative routing* puts the control over the routing process in the hands of the source of the message and works as follows. A source S picks the neighbor A closest to the destination and sends a *next hop request* to A asking for A's neighbor that is closest to the destination. Suppose that A responds with B, then S in turn sends the next hop request to B asking for B's neighbor closest to the destination and so on until the destination is reached. Because of the round trips between the source and the nodes on the routing path, iterative routing increases the latency, it is on average up to 60% higher than recursive routing according to some measurements [102] and since more messages are exchanged bandwidth consumption is also higher [103]. However, by centralizing the control over the progression of routing, *iterative routing* is more fault tolerant, especially in its parallelized form (e.g., in Kademlia [37]) when the source S instead of sending the request for next hops to only one of its neighbors, it sends the request to k neighbors and at any time maintains k outstanding next hop requests to different peers until the destination is reached. We contrast the performance of our approach with parallelized iterative routing in §4.5.

Another solution to message loss in overlays is multipath routing. Messages are routed along many, possibly node-disjoint paths to lower the loss probability and lower the delay [104, 105, 46]. In our system, failures are actively detected and routed around using only a single path. In §4.5 we compare the performance of our system with the routing path diversification approach from [46].

Klemm et al. [106] propose a congestion control scheme which similarly to our work uses the concept of routing around and signaling through feedback. However, our focus is not on avoiding congestion and application rate-limiting, but on detecting and preventing message loss and delays, of which congestion is one of the causes.

The concepts of trust and reputation have been used to detect and then isolate the faulty or selfish peers in non-cooperative environments [107, 108, 109]. Most of this work considers the problem of selecting a reliable peer to perform a transaction with, e.g., a file transfer. Up to our knowledge there has been no direct application of the trust and reputation concepts to fault-tolerant overlay routing, so there are no grounds for comparison with our approach.

The stigmergic routing protocols are inspired by ant colonies and their ability to form robust short paths between the nest and the food sources. In AntNet [110], routing is divided into two phases. The forward ant first attempts to reach the destination. Once the destination is reached a backward ant returns on the same path as the forward ant depositing pheromone at each node which tells the subsequent forward ants in which direction is the destination. Our protocol uses a similar design pattern. When the routing path is traversed, it is either reinforced or weakened depending on weather the delivery to the destination was successful or not. However, unlike AntNet, which is designed for fixed networks, our protocol specifically addresses the dynamic nature and larger scales of the peer-to-peer overlays.

A separate body of work deals with minimizing the delay and stretch in overlay routing. A number of techniques have been proposed ranging from proximity neighbor selection [111] and latency prediction with [112] and without [113] network coordinates. In the solution proposed in this chapter we do not minimize the latency but maximize the fraction of messages delivered under a specified timeout. Existing latency-minimization methods can be independently incorporated into our system to further improve performance.

An early evaluation of the Forward Feedback Protocol (§4.3.2) used in this chapter has been performed in [114]. In the current work, we replace the complex learning algorithm with a more efficient and robust success estimation technique (§4.3.3). We apply the Forward Feedback Protocol to solve a specific fault-tolerance problem and the protocol's effectiveness is demonstrated in an Internet deployment.

4.3. The protocol

4.3.1. Service provisioning

We assume an overlay that offers some *service* that is addressable in the peer ID space. For example in the distributed hash tables (DHTs), key-value pairs are stored and retrieved by specifying the location of the key in the peer ID space. When a peer (the source) requests a service it sends a *service request* to some destination in the peer ID space. We assume that even though the service request specifies a unique location in the ID space, the requested service can be provided by more than one peer (e.g., through key replication in DHTs). The service request is routed hop-by-hop until it reaches the service provider (i.e., the destination).

The source, after it has sent the service request, expects to receive a *service response*. In a correctly functioning overlay the response always arrives. Even in the case when the source asks for a non-existent service, a negative response is sent back to the source. If the response does not arrive within a timeout, this indicates an overlay routing failure and may happen when either the request is forwarded too slowly or is entirely lost. We next define a protocol that informs all the peers on the routing path that a routing failure has occurred. By using that failure information, request forwarders can improve their routing decisions and route around faulty peers.

Our protocol is entirely agnostic to the details of the service response. Service provisioning may be just a single message from the destination back to the source or it may be a complex exchange of messages involving the source, the destination and possibly other peers. The only assumptions we make are (i) that the source can determine the success or failure of service provisioning and (ii) that any service provisioning completes within the T_s timeout. Applications are expected to adjust T_s to their needs.

We define the *lookup* to be the whole process of issuing and routing the service request until the service response is received.

4.3.2. Forward Feedback Protocol (FFP)

The *Forward Feedback Protocol* (FFP) is to disseminate the routing failure information to all the peers on the routing path. There are two outcomes possible when routing a service request: either success, when the source receives the service response or failure when the source does not receive the response within the timeout T_s. The source is the only peer that can determine the outcome of the routing process. When the outcome is known the source sends a *feedback message* that is recursively routed along exactly the same routing path as that taken by the service request (Fig. 4.1). The feedback message is either *positive* or *negative* depending on whether the outcome was a success or a failure. As the peers along the routing path receive the feedback they learn whether their routing decisions led to the successful service provisioning.

Each peer on the routing path other than the source or destination after forwarding the service request expects to receive a feedback message from the previous hop. All service requests and their corresponding feedback messages contain the same unique identifier. When a service request is received a node allocates the state containing the unique identifier from the incoming the service request and the address of the sender. This state is used to match the incoming feedback messages with the previously forwarded service requests and to forward the feedback backward on the path. The state is kept at a node i until either: (1) the feedback arrives at i or (2) the service response arrives at i and i is the source or (3) T_s fires at i and i is the source or (4) T_f fires at i and i is not the source. The $T_f > T_s$ is a timeout specifying how long a

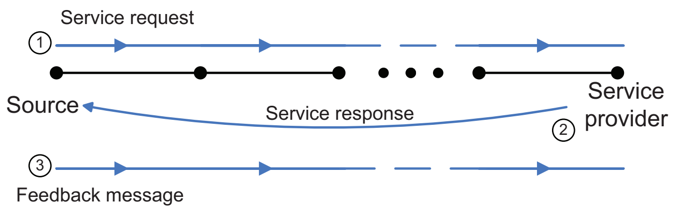


Figure 4.1. The Forward Feedback Protocol (FFP). The service request is recursively routed in the overlay until it reaches the service provider (the destination). The destination sends the response back to the source. When the service response is successfully received or the T_s timeout fires the source sends a feedback message along the same routing path that the service request took. After receiving the feedback, the peers on the routing path learn whether their routing decisions led to the successful lookup.

non-source node waits for arrival of the feedback message.

4.3.3. Fault-tolerant routing

Each peer locally keeps a set of *success estimators*. The success estimators reflect the history of the past lookup outcomes. When a service request arrives and needs to be forwarded the peer selects the next hop for which the success estimate is the highest. The arriving feedback messages are used to update the success estimators. As the peer is forwarding service requests and receiving feedback it improves its routing decisions.

Success estimators. Assume that some peer i has n neighbors to which it can send service requests. Let the identifier of i be p_{ID}. The neighbors are indexed by $j = 1 \dots n$. The peer i keeps one success estimator $\hat{\theta}_{j,d_z(d_{ID})}$ for every (neighbor, destination zone) pair. The *destination zone* $d_z(d_{ID})$ is the function of the destination ID d_{ID} defined as $\lfloor -\log_2 d(p_{ID}, d_{ID}) \rfloor$, where $d(p_{ID}, d_{ID})$ is the distance in the ID space between p_{ID} and d_{ID} specified in the service request. The properties of the zoning function and the rationale behind it are illustrated on Fig. 4.2.

The success estimator $\hat{\theta}_{j,z}$ kept by node i models the neighbor j's reliability of delivering service requests to zone z. Each $\hat{\theta}_{j,z}$ is represented by two numbers a and b which are the exponential moving average counts of the observed positive (a) and negative (b) feedback. The value of $\hat{\theta}_{j,z}$ is simply defined as $\frac{a}{a+b}$. The a and b counts are initialized to 1. When positive feedback arrives the updates are: $a \leftarrow \gamma a + 1$, $b \leftarrow \gamma b$, on negative feedback: $a \leftarrow \gamma a$, $b \leftarrow \gamma b + 1$.

The parameter $0 < \gamma < 1$ provides a single knob for controlling how sensitive the system is to failures. Low values of γ result in a system that readjusts its routing paths quickly in response to failures but the routes are less stable since even a few lost messages can affect them. In a high γ system more feedback needs to accumulate before the peers change their routing decisions but the decisions are more reliable.

Next hop selection. Given a service request r with a destination d_{ID}, the node

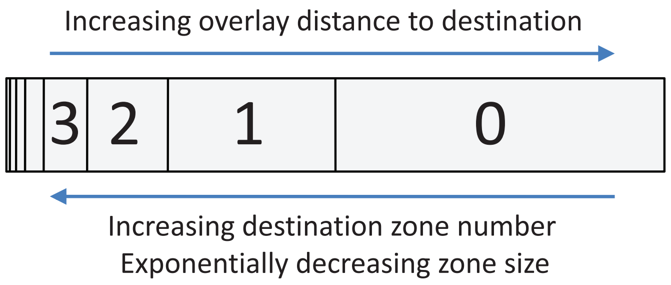

Increasing overlay distance to destination
3
2
1
0
Increasing destination zone number
Exponentially decreasing zone size

Figure 4.2. Destination zoning. Zoning is key to FFP's scalability. Rather than keeping per-destination success estimators which would not scale we divide the destination space into zones and only keep per-zone success estimators. The further the zone is from the peer the larger the portion of of the destination space it covers. Since the zone sizes decrease exponentially there are $O(\log N)$ of them in terms of the network size, which dramatically reduces the protocol's local state size.

i selects such next hop h that the predicted success probability is maximized, i.e., $h = \arg\max_{j \in 1 \ldots n} \hat{\theta}_{j,d_z(d_{ID})}$, where $j \in 1 \ldots n$ are all the neighbors of i in the overlay.

While forwarding the request each node decrements the request's TTL (time-to-live) counter. When it reaches zero the request is dropped.

Self-organized fault-tolerance. The next hop selection rule makes the service requests follow the most reliable paths. When the lookup fails all the peers on the routing path decrease their success estimates for their downstream neighbors on the path, the routing path is weakened. In the same way, when a lookup succeeds, the path is positively reinforced and more likely to be used in the future. As lookups are performed, each node locally and independently accumulates reliability information about its overlay neighbors. The network as a whole converges on reliable routes in a completely decentralized self-organized way. This process is continuous and when delay or loss conditions change, the network detects that and reroutes the traffic to increase reliability (Fig. 4.3).

Topology-oblivious routing. In contrast to existing overlay routing protocols, FFP's next hop selection does not rely on the overlay neighbors' peer identifiers but only on the neighbors' reliability history. In that sense, FFP is entirely oblivious to the topology of the network it routes in. This is key to FFP's universal applicability to a wide range of topologies.

Consider an overlay that has never forwarded any service requests. Initially all success estimators on all peers are $(a = 1, b = 1)$. Due to the properties estimator updates and the next hop selection the service requests either tend to visit the next hops that have not been visited before or those that have already been successful at delivering to a destination zone. Given enough service request traffic the system eventually by trial and error learns which next hops are best for which destination zones and routing becomes efficient. Despite starting with no knowledge about the next hops at all, this process converges relatively fast (§4.5.3). Successful routing paths are rapidly reinforced and service request traffic is redirected to use them.

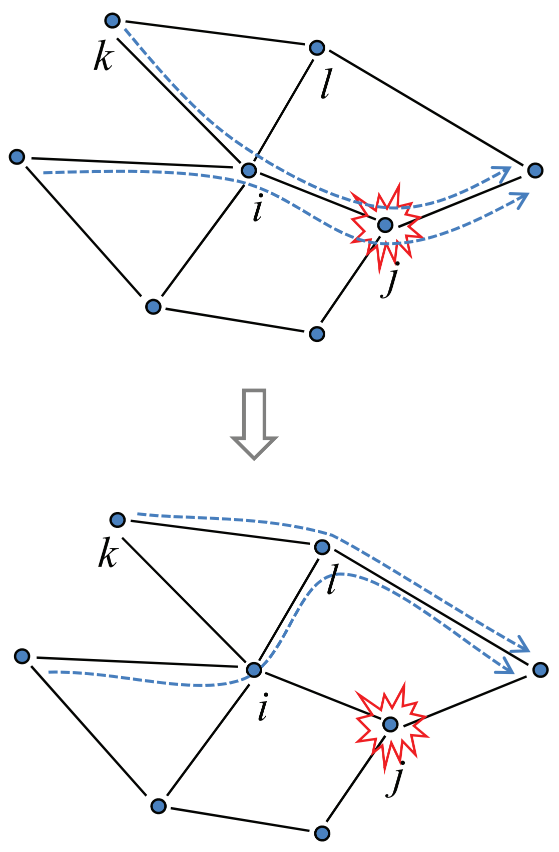


Figure 4.3. Self-organized fault-tolerant routing. Two routes are passing through peer j. When j starts failing the two routes are weakened by negative feedback. Peer i starts associating failure with its neighbor j and switches to l as the next hop. Similarly, node k routes around i, however, in this k's next hop i is not faulty, it only happens to lay on a faulty route. In general, it does not matter which node on the routing path reacts to failure as long as some node does. Both i and k made their decisions to reroute independently and based only on local observations. This illustrates how an FFP-based network of independent nodes self-organizes and finds reliable routes.

Optimized bootstrap. Initially, when no feedback information is available the peers make essentially random routing decisions. Over time, as more successful paths are found and reinforced with positive feedback the routing becomes reliable for all source-destination pairs. This process can potentially be slow, but only takes place when a large group of uninitialized peers suddenly joins the system, which is a rare case in practice. However, to account for this we introduced a performance optimization: the *warmup phase*. The warmup phase starts after the peer joins and ends when the peer has received at least $f_{min} = 100$ feedback messages. During the warmup phase the peer uses the usual greedy distance-minimizing rule for routing in the overlay. When the warmup phase has finished the peer switches to FFP routing and starts using the feedback gathered during the warmup to make routing decisions. We found the simple $f_{min} = 100$ rule to perform well in all the workloads and failure scenarios used in the evaluation (§4.5).

Loop-freedom. Most of the existing protocols follow the greedy approach in which each hop attempts to minimize the distance to the destination in the peer ID space. FFP routing is driven by the success estimators and not the ID space distance. To increase the routing reliability the peers in our system deviate from the greedy routing paths. Because of these deviations, one desirable property of greedy routing is lost: loop-freedom. However, routing loops are extremely rare in FFP. There are two reasons for that. First, any loops that occur lead to exhaustion of the service request TTLs, which leads to failures and negative feedback. The negative feedback is then received by the peers that are part of the loop and they start using alternative routes most likely breaking the loop. Second, when loops occur they increase the lookup latency, which leads to the T_s timeouts followed by negative feedback and the breaking of the loop.

Achieving near loop-freedom without explicitly enforcing it is a good illustration of FFP's path-wide fault tolerance in action. It does not matter how the service requests are routed as long as they do not exhaust the TTL and the response arrives at the source within T_s. Any routing inefficiencies such as loops are eliminated by peers changing the routing paths in response to negative feedback.

4.4. Scalability, overheads and optimizations

Each peer needs to store the success estimators. Assume a system of N peers and some peer i with d neighbors. The destination zoning function divides the whole set of peers into $O(\log(N))$ zones. Hence, there are $O(d \log N)$ $\hat{\theta}_{j,d_z(d_{ID})}$ success estimators. Assuming $d = O(\log N)$, the total state size for the success estimators is $O(\log^2 N)$, which scales well with the network size. A peer must also keep the state between arrival of the service request and either receipt of the corresponding feedback message or the T_f or T_s timeout. Assuming the rate λ of incoming service requests and $T_s < T_f$, the expected state size is $O(\lambda T_f)$, which in practice does not pose a problem since request rates in P2P systems are low and overlays are typically well load-balanced in terms of forwarded traffic.

The bandwidth overhead in our system consists of the feedback messages and the unique identifier field that needs to be added to the service requests. A feedback message consists of the unique identifier and a Boolean flag indicating either success or failure. The naive approach is to send each feedback message separately, but this can be optimized in a number ways. First, feedback can be piggybacked on other service requests sent to the same neighbor. Second, the largest part of the feedback message is the unique identifier. If the identifiers are assigned sequentially and independently for each ordered pair of connected peers, then feedback can be sent for contiguous ranges of identifiers, which can simply be specified by their left and right bounds. If the ranges are big enough the total amortized bandwidth overhead is reduced to a single bit per feedback message.

4.5. Evaluation

The system is implemented using the ProtoPeer toolkit (§A) and evaluated in a 400-node PlanetLab[1] deployment.

We consider the following routing protocols: (1) **BASE** - the baseline without any routing fault-tolerance mechanisms, (2) **MULTIk** - multipath routing as in [46], in the first hop the source chooses k neighbors closest to the destination instead of one, (3) **ITERk** - iterative routing scheme based on Kademlia [37] with k simultaneous lookups, we set a low timeout of 500ms on the hop-by-hop responses so that peers can recover from local failures and resume routing on-the-fly and (4) **FFP** - system running the FFP.

We use a bidirectional Chord [36] implementation. Each peer sends one service request for some key every 500-1500ms. We create a set of 1024 keys uniformly randomly distributed in the ID space. The service request keys are drawn from this set according to a power-law distribution such that 50% of the requests are for the top 5% of the keys. This constitutes a realistic workload for a large number of P2P applications [115].

The request is considered delivered when it reaches either the peer responsible for the key or one of the two of the peer's immediate Chord successors or predecessors, i.e., in the DHT terms, there is a 5-fold key replication. The timeouts are $T_s = 3s$ and $T_f = 6s$ and the TTL is set to 20. Note that T_f applies only to FFP, while T_s and TTL apply to all four routing protocols. A lookup fails when either the TTL is exhausted or T_s fires before the service response is received by the source. FFP's γ parameter (§4.3.3) is set to 0.95, which offers a good balance between the sensitivity to failures and the tolerance to feedback noise.

Each deployment is approximately 400 PlanetLab hosts in size depending on the availability. The standard practice in PlaneLab experiments is to exclude highly loaded hosts from the deploy list. We have included all the hosts that we could log on to provide a more challenging environment for evaluating the fault-tolerance. The hosts were polled for their 15-min UNIX load averages. The median was 5.11 and the 90^{th} percentile was 16.12. Under these conditions CPU starvation was causing considerable delays on some peers.

4.5.1. Message loss

To simulate peer overload some fraction of peers are droppers, which drop messages with probability 0.5. Enough messages get through such that the neighbors of the droppers do not consider them as departed from the network and keep them as their overlay neighbors. Just as the other peers, the droppers periodically send their own service requests, but they might get dropped on sending as any other message.

[1] http://www.planet-lab.org/

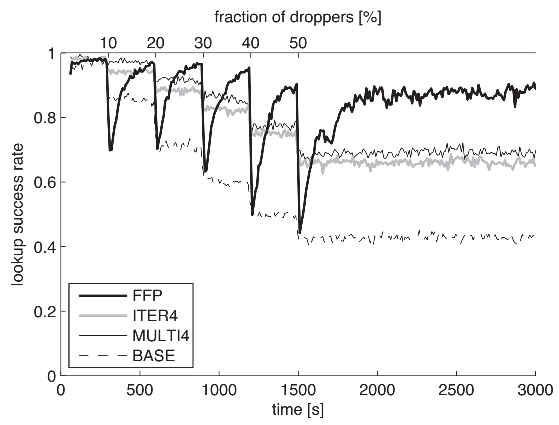


Figure 4.4. Resilience to message dropping. Every 5mins. a new 10% batch of peers starts to drop all messages except their own lookup traffic. FFP responds to the failures and routes around the droppers, which allows it to reach the delivery success rates up to 30% higher than the existing approaches.

In our failure scenario, every 5 minutes a new 10% batch of peers becomes droppers adding to the existing set of droppers. New droppers stop arriving when half of the peers become droppers.

Success rate. We measure the lookup success rate, i.e. the number of service responses that arrived before the T_s timeout as the fraction of all service requests that have been sent by the sources. Figure 4.4 summarizes the results.

As more droppers arrive the lookup success rate decreases sharply for the system without any routing fault-tolerance mechanism (BASE). The multipath (MULTI4) and iterative (ITER4) routing approaches improve the lookup success rate by 30%. An FFP-based system, when it converges, is able to achieve a high 90% success rate despite the fact that half of the peers in the system are dropping messages.

The droppers do not send any feedback messages even though they are the sources of service requests. Although there is less feedback exchanged in the network, the available feedback provides the peers with enough information to effectively adjust the overlay routes.

Bandwidth usage. In the dropper experiment we have also measured the total number of bytes that have to be transmitted by the peers per successfully completed lookup (Fig. 4.5).

Both the multipath and the iterative routing approaches rely on redundantly sent messages to tolerate failures, which considerably increases the bandwidth cost. ITER4 is worse than MULTI4 in two ways. First, every time an RPC times out in ITER4 it is retried, which in presence of many droppers results in many retries, which increase the bandwidth cost considerably. Second, because of the round trips back to the source the total lookup latency is much greater than in the case of MULTI4 and many ITER4 lookups take longer than T_s and fail (Fig. 4.4).

Although we have not investigated this, our implementation can be further optimized

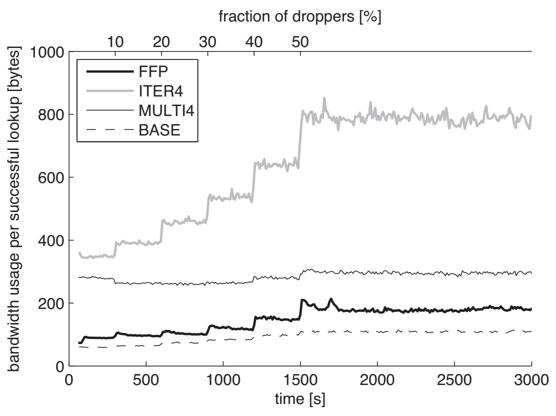


Figure 4.5. Per-lookup bandwidth usage. We sum up the total send and receive bytes from all the peers. This value is then divided by the number of successful lookups and plotted. MULTI4 routing relies on 4-way message redundancy for fault-tolerance, which results in 4 times the bandwidth cost compared to the baseline. ITER4 has a 1s timeout on the RPCs and when many peers are dropping messages there are many RPC retries which increases the bandwidth cost considerably. FFP's only overhead are the feedback messages (§4.3.2), which add up to only a marginal bandwidth increase compared to the system without any routing fault-tolerance mechanism (BASE). As the droppers arrive in the system, FFP routes around them, this increases the path lengths and thus the per-lookup bandwidth cost.

by piggybacking the feedback messages on other messages or sending feedback messages in batch (§4.4), thus avoiding message header overhead, which is the dominating component in case of the simple feedback messages.

4.5.2. Tolerance to message delays

We turn to evaluating the tolerance of our system to peers delaying the messages. Apart from the inherent PlanetLab delays we introduce artificial ones. Similarly to the case of droppers a new 10% batch of peers becomes delayers every 5mins. Each delayer delays all messages by an interval uniformly randomly chosen from between 100 and 2000ms.

Figure 4.6 shows the lookup success rate. Multipath routing with its four independent routing paths is able to reach the destination at least with one of them. Iterative routing fails entirely in this case. When the sources are also the delayers they significantly slow down the whole iterative routing process. Due to the high delay variability FFP needs 500s to gather enough feedback information to route around the delayers. Once it converges it reaches the same high lookup success rate as the multipath routing.

We have also measured the median lookup latency for the successful lookups (Fig. 4.7). The multipath routing method besides increasing the fault-tolerance to delays also lowers the lookup latency. Our system is designed to maximize the fraction of lookups completed within a timeout $T_s = 3s$ and not the lookup latency median. However, as a side effect of rerouting

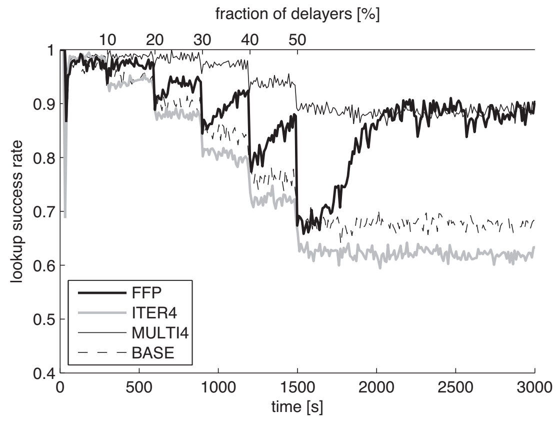


Figure 4.6. Tolerance to message delays. Every 5mins a new 10% batch of peers starts delaying all outgoing messages by 500-2000ms. The multipath algorithm performs well, while the source-controlled iterative routing is significantly slowed down by the delaying sources. The feedback that FFP peers are receiving is noisy due to the high variability of the path delay. FFP needs 500s to detect and route around the delayers after the last failure injection at 1500s. When it converges, FFP reaches the high performance level of multipath routing.

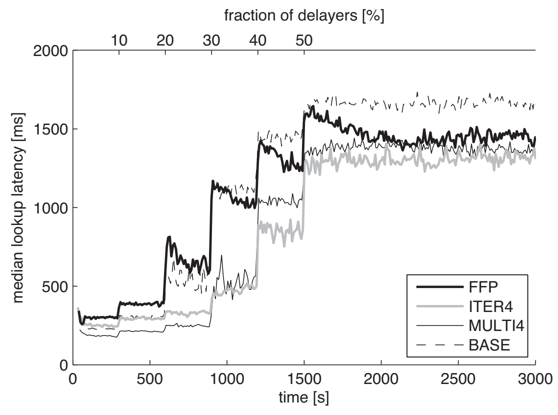


Figure 4.7. Median lookup latency under forwarding delays. The same setup as in Fig. 4.6. Multipath routing with its four independent routing paths achieves the lowest latency. Even though FFP was not designed for latency minimization, it removes slow peers from the routing paths when the lookup latency exceeds the T_s threshold, which over time leads to latency decrease.

for reliability FFP also slightly lowers the median delay.

4.5.3. Topology-oblivious routing & the bootstrap

One of the novel properties of our protocol is its ability to route in overlays while using only the reliability feedback and ignoring any overlay topology information (§4.3.3) such as the

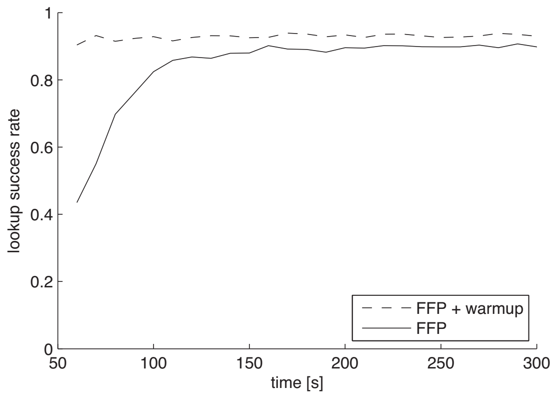


Figure 4.8. Topology-oblivious routing & the bootstrap. Two instances of FFP are compared, with and without warmup (§4.3.3). All the peers' success estimators are initially $(a = 1, b = 1)$. At 60s all the peers start their workloads simultaneously. FFP without warmup converges on efficient loop-free routes relying solely on the reliability feedback and ignoring any overlay topology information. The warmup shortens the initial convergence time as well as slightly improves the overall performance.

locations of the neighbors in the peer ID space. In this section we demonstrate this property experimentally.

All the peers initially have no previously recorded feedback, i.e., all their success estimators are $(a = 1, b = 1)$. At the one minute mark all the peers simultaneously start their workloads. Initially, the routes are explored randomly, there are many failures, but over time each node learns which of its next hops are reliable forwarders for which destinations.

To create a more challenging scenario we use a workload in which peers send service requests to destinations that are at least 0.25 away on the unit Chord ring. Such workload has a larger fraction of long multi-hop routing paths.

We compare the systems with and without warmup. Warmup is an optimization that speeds up convergence by initially relying on topological information and then subsequently switching FFP's purely reliability-based routing (§4.3.3).

The results on Fig. 4.8 shows that in a system without warmup the uninitialized FFP peers are still able to converge on efficient routes despite ignoring any topological information. Moreover, the created routes are loop-free, despite the fact that loop-freedom is not explicitly enforced by FFP. Loops lead to failures and the peers in a self-organized way learn to avoid them.

The measurements also confirm that the warmup process significantly speeds up the initial convergence.

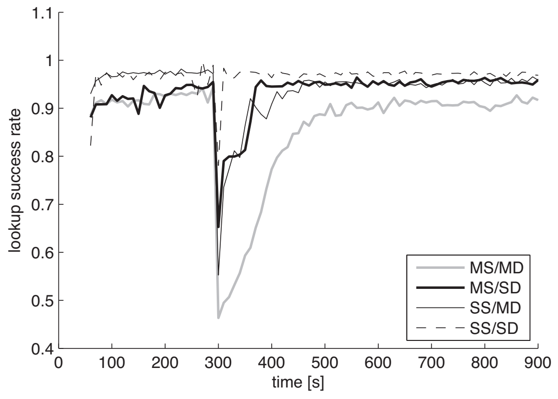


Figure 4.9. FFP performance under different workloads. The four workloads are: multi-source/multi-destination (MS/MD), single-source/multi-destination (SS/MD), multi-source/single-destination (MS/SD) and single-source/single-destination (SS/SD). At 5mins 25% of the peers become droppers (§4.5.1). The fewer paths the traffic is confined to the more feedback the peers on these paths receive and the faster the recovery after the failure.

4.5.4. The influence of the workload on convergence time

Until now our FFP evaluation has been based on a power-law workload, we next explore how the workload influences the FFP's performance. We are next going to consider four workload variants: multi-source/multi-destination (MS/MD), single-source/multi-destination (SS/MD), multi-source/single-destination (MS/SD) and single-source/single-destination (SS/SD). In the single-destination workloads peers issue lookups for a single fixed key. In the multiple-destination workloads destination keys are chosen uniformly at random from the ID space. In the single-source workloads there is a single high-rate peer sending lookups, the other peers are passive. The workloads are normalized such that the lookup rates summed across all the sources in each workload are the same.

Figure 4.9 compares how the system recovers from a failure under different workloads. The general observation that can be made is that the fewer the source-destination pairs in the workload the faster the failure recovery. This is particularly evident in the SS/SD case. When traffic is constrained to fewer paths, the peers along these paths receive more feedback and can adapt faster. Moreover, the failure recovery is the faster the fewer there are destinations in the workload. The peers in the neighborhood of the destinations receive more feedback and with it they can determine which of the destination ID replicas are more reliable.

4.5.5. Tolerance to churn

So far we have evaluated our system in setups with all peers permanently on-line, however in P2P systems peers are constantly joining and leaving the overlay (churn). Peer departures in

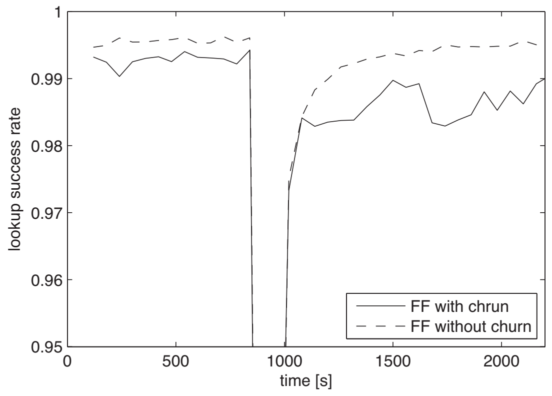


Figure 4.10. FFP performance under churn. An FFP system is running a multi-source multi-destination workload. At 800s 40% of the 1024 peers become droppers. The time resolution is 60s. Feedback message loss caused by churn and the constant joining of new uninitialized peers do not affect FFP's ability to recover from the failure.

our system cause forwarded messages to be lost and also break the feedback flow, which affects the state of the success estimators, which in turn might disrupt FFP's next hop choices. In addition, churn constantly adds new peers which initially are in the warmup mode (§4.3.3) and make routing decisions that do not account for the lookup failures. In the following experiment we test to what extent these effects affect the system's performance.

We take the same application code as used in the PlanetLab deployment and evaluate it in the ProtoPeer simulator. The Internet latencies are taken from the King dataset[2]. We are using the churn model based on the Kad4 data from the Stutzbach's et al. study [116], which measured churn characteristics in live P2P networks. There are 1024 live peers on average. At 800s into the simulation, 40% of the peers become droppers. The results of the simulation with and without churn are compared on Fig. 4.10. The ability to detect and route around failures is largely unaffected by churn. There is only a $1-2\%$ drop in the lookup success rate, but this is mainly caused by the unpreventable message loss due to the peer departures.

4.6. Discussion

Self-organized fault-tolerance. In the Forward Feedback Protocol each node locally and independently of the other nodes learns to route based on the simple binary feedback. As we have confirmed in the evaluation, an overlay composed of independent FFP nodes continuously self-organizes to increase the routing reliability. When message loss (§4.5.1) or excessive delays (§4.5.2) occur in the network, the paths are rapidly readjusted to route around the faulty peers and overlay links.

[2] http://pdos.csail.mit.edu/p2psim/kingdata/

Bandwidth efficiency. The performance data accumulated at each peer about its neighbors is key to FFP's reliability. The other approaches do not keep track of their neighbors' performance and instead rely on message redundancy to deal with the failures. As we have confirmed in the measurements this comes at a significant bandwidth cost. Knowing the neighbors' performance allows FFP to choose more reliable paths which allows our protocol to reach higher success rate than the existing approaches.

However, despite the high bandwidth cost, the usage of redundant routing paths in iterative and multipath routing allows for reduction in the median lookup latency. In contrast, FFP is better suited for bandwidth-constrained environments and applications in which latency minimization is not the highest priority.

Convergence time & suitability for high-rate traffic. Convergence time in FFP is directly dependent on the rate at which the peers are forwarding the traffic. The higher the the traffic rate the more feedback is available and the more reliable the routing decisions are. As we observed (§4.5.4) convergence is significantly faster when the traffic is less diverse, with fewer source-destination pairs . The peers are able to respond to changing conditions faster since the feedback accumulates in a smaller set of success estimators. The fact that the more peers forward the quicker they converge on reliable routes as well as FFP's low bandwidth overhead make this protocol particularly suitable for applications with high-rate traffic.

Combining the routing approaches. So far we have looked at our solution to fault-tolerant overlay routing as an alternative to the existing approaches. However, an interesting possibility is the integration of our solution with the multipath or iterative routing protocols. Every time the next hop is chosen in one of these protocols, it is chosen greedily to minimize the distance to some destination ID. This greedy next hop choice can be replaced with our failure-aware choice (§4.3.3) taking the feedback into account. This would combine the fault-tolerance gains from both the message redundancy of iterative and multipath routing and the failure-awareness of our routing approach.

Under abundant bandwidth the system could reap the full benefits of all approaches, when bandwidth became scarce the system could reduce the amount of message redundancy (the k parameter of MULTIk and ITERk) or entirely fall back on routing only with our bandwidth-efficient method.

Feedback suppression & incremental deployment. In FFP a service request is always followed by a feedback message. But does it have to? We have indirectly measured that moderate feedback loss does not prevent FFP from converging on reliable routes (§4.5.1). This brings up an interesting question: How much of the feedback is necessary for reliable routing? The system could suppress feedback generation to conserve bandwidth. It would be possible in such a system to precisely control the tradeoff between reliability and the amount of bandwidth invested in maintaining reliable routes.

If the system can tolerate feedback dropping then it will tolerate a fraction of peers that do not support the FFP protocol and ignore the feedback messages instead of forwarding them.

This opens up the possibility of incremental deployment of FFP in already running overlays, which could proceed as follows. Initially, the isolated FFP peers do not receive any feedback and rely on the traditional overlay routing. As more FFP-compatible peers are deployed and start forming larger connected components in the overlay topology they begin exchanging feedback, soon they accumulate enough feedback to exit the warmup stage and begin routing using their success estimators.

Unstructured vs. structured vs. FFP overlay routing. Unstructured overlays (e.g., [92]) do not assign any identifiers to the peers and do not maintain any global overlay topology. Instead of routing they rely on random walks or flooding for message delivery. Similarly to the unstructured overlays, the FFP peers do not need to know their neighbor's identifiers to be able to route. Also, just as the unstructured overlays, FFP can operate in the topology-oblivious mode in which it finds the destinations using random walks, however unlike the unstructured overlays, FFP learns from its routing mistakes and the random walks are over time replaced with more reliable faster paths.

Even though FFP makes no use of the neighbor identifiers each peer locally uses its own identifier as the input to the zoning function (§4.3.3). The zoning function makes the assumption that the destination IDs close to one another in the ID space are also likely to be topologically close in the overlay graph. This is a property shared by all structured overlays [36, 35, 88, 89, 39] and allows FFP to scale.

Richer signaling. The proposed feedback mechanism is general enough to be used in applications other than overlay routing fault-tolerance. Feedback messages can be used to signal any Boolean property of the routing path, not only when it satisfies the T_s latency constraint. Forwarders can route around those peers that consistently cause the property of interest to become false. For example, when service requests pass through forwarders or service providers close to reaching their throughput saturation point, these peers can signal it by setting a congestion control bit in the service requests (as in [106]). The source then finds out about the congestion via the service response and sends negative feedback along the path to notify the forwarders that some peers on the path are close to being overloaded and should be avoided.

Feedback can be used to signal not only the Boolean properties of the routing path but others as well. For example, if feedback messages contain the delay information, then the success estimators can be modified to be delay estimators and the system can potentially be turned into a latency-minimizing overlay [113, 112].

Beyond overlay routing. We have applied FFP to solve the problem of fault-tolerance in recursively routing overlays. However, FFP is general enough to be layered on top of any recursively routing network. For example, we have already obtained several promising results from applying FFP to multi-hop wireless networks. In such networks, FFP provides both the routing functionality and the route discovery functionality, which uses FFP's random walks to locate the destinations. In wireless networks we do not need to rely on zoning (§4.3.3) for scalability, success estimators can simply be stored per-destination. In general, FFP can be adapted to the needs of a specific application or a network by modifying the granularity of the success estimators and assigning some meaning to the Boolean success flag carried by the feedback.

In Chapter 5, we adapt FFP to the use in mobile ad-hoc networks by letting the feedback flow not forward, but backward from the destination back to the source. In addition, instead of using random walks the random walks for route discovery, we take advantage of the broadcast nature of the wireless medium to flood the network with route requests.

Security. The protocol proposed in this chapter rapidly adapts to node arrivals and departures, intermittent node failures and delays. The overlay running FFP will still be resilient to these effects if they are intentionally caused by an adversary. However, there exist several vulnerabilities that an adversary can exploit. Perhaps, the more serious one is the possibility of false feedback messages. If an adversary sends false positive feedback messages and at the same time drops the lookup messages, then the nodes will continue to route through the adversarial route as if it was correct. We address this and other security problems in the next chapter by constructing an authentication scheme in which each intermediate node can verify the origin and integrity of the feedback messages.

4.7. Conclusions

We have presented a solution to the problem of message loss and excessive delays during lookups in structured overlays. We have shown how nodes acting completely independently can self-organize into a reliably routing overlay using only the binary feedback messages as the signaling mechanism. We have confirmed the effectiveness of our solution in an Internet deployment under a variety of failure scenarios and workloads. The FFP protocol has up to 30% higher delivery success rate than the existing approaches while using 2-5 times less bandwidth.

Our protocol can readily be integrated into most of the modern structured overlays to provide fault-tolerance at low bandwidth cost. For example, FFP could be used in conjunction with the overlay network formation protocol that we proposed in the previous chapter (§3). This would give the nodes a complete autonomy in making their routing and connectivity decisions, which in turn would allow the nodes to more flexibly manage their own resources as well as more fluidly adapt to the changes in the overlay resources made available to them.

For additional fault-tolerance, FFP can be combined with either the iterative or multipath approaches. However, fault-tolerance is not the only area of application. FFP can be viewed as a general signaling protocol and can be used for overlay congestion control or lookup latency minimization. Because of its general specification, FFP can be used to improve routing fault-tolerance other networks than peer-to-peer overlays. In the next chapter, we adapt the FFP protocol to routing in mobile ad-hoc-networks.

Publications: [117, 118]

Chapter 5

Castor: a Scalable Secure Routing Protocol for Ad-hoc Networks

5.1. Introduction

While the Forward Feedback Protocol from the previous chapter is robust to message loss and delays, it is vulnerable to malicious message injection. More specifically, the attacker could send fabricated positive feedback messages convincing the nodes downstream on the routing path that the delivery is successful, while at the same time dropping the messages that are being routed. In this chapter, we address this and other security problems by proposing an adaptation of the Forward Feedback Protocol to the mobile ad-hoc networks (MANETs).

The peer-to-peer, distributed operation of MANETs, as well as the nature of wireless communication pose significant security challenges (§2.3). Without appropriate security mechanisms, the adversary can affect or even control the self-organizing operation of the network, and degrade or completely prevent communication (§2.3.2). Thus, a fundamental security objective in such systems is to *secure communication.* Secure communication protocols should be able to maintain acceptable data delivery rates under all feasible attacks. *Secure route discovery* is essential for solving this problem. Without it, the data sent across the routes the adversary manipulated would never be received at their destinations. However, securing the route discovery is not sufficient: adversaries can always become a part of the routes by behaving correctly during the route discovery, and then strategically disrupt communication (e.g., drop or corrupt packets) once the routes are being used.

It is thus necessary, as some protocols surveyed in §5.2 do, to utilize a *secure data transmission* protocol on top of secure route discovery. Secure data transmission protocols correlate data delivery failures with specific routes or network areas, possibly controlled by the adversary. Then, they reroute the traffic, to avoid the adversary and reestablish reliable communication.

This approach was shown to be effective, especially if sufficiently rich connectivity information is available. Simultaneous use of multiple paths, redundancy in transmissions, and end-to-end secure feedback allow for quick route convergence [119]. However, because of system constraints, there may not be enough bandwidth available for multi-path data transmission. When resources are scarce, the generally applicable solution is a single-path secure communication protocol: sending data and feedback across a single path, and switching to another path once the current one is deemed unreliable.

Would such a solution remain efficient and effective in large and highly volatile networks, even in the presence of powerful adversaries? Consider networks that are open, mobile and can grow in size, with the subset of nodes supporting a given single-path flow constantly changing. Managing the variability, avoiding the faulty and adversarial nodes, while sustaining reliable communication is a challenge.

Our *Continuously Adapting Secure Topology-Oblivious Routing (Castor)* addresses exactly this problem. Each node keeps track of the reliability of its neighbors only; none of the local state is ever exchanged over the network; packet sizes do not carry routes, thus their length does not grow with the network size; each node operates fully autonomously, making routing decisions independently of other nodes and without knowing the network topology beyond its local one-hop connectivity. These features make Castor *scalable*. Moreover, in-network routing state allows Castor to *rapidly adapt* to a wide range of faults, malicious and benign, even under an *overwhelming adversarial presence*. Finally, the minimal exchange of information between the nodes implies there is little need for authenticating it or securing its transmission; this enables Castor to operate under the *simplest*, among those in the literature, *trust assumptions*.

Our extensive comparative evaluation shows that Castor outperforms four other protocols (SRP/SSP, Sprout, SEAD, and trivially the non-secure AODV), with significant advantages:

- Castor consistently achieves up to *40% higher packet delivery rate without any additional overhead*
- It recovers at least *twice as fast* as the other protocols
- It is the *only one* among the five evaluated here that achieves *full recovery* from the wormhole attack without the help of a secure neighbor discovery protocol.

Equally important, Castor maintains its advantage as the network *scales* and mobility increases: for example, in a 400 node network with 80 black-holes and continuous mobility, Castor achieves, with a mild overhead increase, a consistent 60% packet delivery rate, *double* that of other protocols.

What sets Castor apart is its fundamentally novel approach, among secure communication protocols, and its versatility, in spite its simplicity. Castor is the first protocol to demonstrate robustness against such a large spectrum of attacks and for a wide range of network scales. Our comprehensive comparative evaluation of secure communication protocols is also the first of its kind.

In the rest of the chapter, we first discuss related work (§5.2) and give an overview of Castor. We then define the system and adversary models (§5.4) and present in detail the functionality of Castor (§5.5). We analyze its security (§5.6) and evaluate the performance (§5.7) before we discuss open questions (§5.8) and conclude.

5.2. Related Work

Secure ad-hoc networking protocols address two main issues: (a) secure route discovery, to prevent attacks on the disseminated routing information, and (b) secure data transmission, to ensure data delivery. Most proposals in the literature considered the first issue only or assumed the second one is addressed by upper layer protocols; few ones considered both issues.

Secure route discovery. SRP [120] is an on-demand protocol: it floods in a controlled manner a route request (RREQ), with intermediate nodes each appending its identifier. The destination returns route reply (RREP) packets strictly across the reverse of the path accumulated in the RREQs. End-nodes can authenticate each other and their RREQ and RREP packets; intermediate nodes do not need to authenticate traffic from end-nodes. Ariadne [121] follows the same principle, but it authenticates intermediate nodes at the end-nodes. This increases the trust management complexity and overhead, in return for stricter identification of the intermediate nodes at the source. endairA [122] takes essentially the same approach, utilizing only public key cryptography, and offers increased resilience to attacks.

SRP, Ariadne, and endairA provide the entire discovered route (connectivity information) to the source node, and the same is true for link state protocols such as SLSP [123]. In a different category, implicit route discovery protocols [124] provide each node with the next hop towards the destination: ARAN [125] discovers a single route, based on the first-returning RREP at the source; S-AODV [126] provides security for AODV [127], authenticating its RREQ, RREP and route error packets; and SEAD [128] protects distance-vector calculations from distance decrease, using symmetric key cryptography (while ARAN and S-AODV use digital signatures).

Secure data transmission. SSP [129] is a secure single-path protocol that relies on an end-to-end security association; it transmits packets across a route calculated over the connectivity the underlying route discovery provides (typically, protocols such as SRP). The destination validates received data and responds with acknowledgements; if not, the source detects a packet loss. The route rating is increased each time an acknowledgement is received, and it is reduced when a timeout occurs (no ACK); once the rating drops below a threshold, the route is discarded and the source switches to another one (invoking a new route discovery if needed). SSP is robust to any attack (e.g., wormholes, tunnels, other collusion attacks) that causes a packet to be dropped; if so, the route is discarded.

Sprout [130] is a protocol that source-routes data across a single path chosen among many alternative ones. Those paths are calculated over the topology view a secure link state discovery protocol offers, with nodes broadcasting link state updates across the network. In order to be resilient against colluding adversaries that advertise fictitious links, Sprout introduces mechanisms that prevent the pollution of the network link state view. Routes are generated and utilized probabilistically, acknowledgements are returned by the destination, and routes deemed operational are re-used while new alternative ones are explored. The link-state operation requires that any node can identify all other nodes at all times. SSP and Sprout are the two protocols closer to our Castor.

Other related schemes. ODSBR[131] discovers routes reactively, it updates link weights based on their behavior observed at the sources, and when communication reliability drops below a threshold, it augments data packets with probes to identify the wrong-doer. ODSBR requires that the source knows all nodes in the network. It can maintain reliability across a route above a threshold unless two or more colluding attackers are part of the route [130]. Beyond security protocols, reputation, and remuneration-based schemes have been considered [55]. All these schemes are complex and costly (e.g., requiring a full trust graph, long observation periods), or they can be effective only against rational adversaries, or they can be susceptible to attacks that incriminate correct nodes. Finally, a note on the so called ant-based routing protocols [110, 132, 133] which somewhat resemble Castor: they do not have an explicit route discovery phase and use the "acks" to positively reinforce paths (by analogy to phermone traces). However, none of them considered security.

Comparison to Castor. In brief, Castor extends over secure route discovery, as it falls in the category of the comprehensive solutions that secure data transmission too; e.g., ODSBR, SRP plus SSP or SMT, and Sprout. Compared to those, it introduces significant differences. Concisely: (i) Routes need not be attached to packets, thus packets do not grow in length with the network size (and thus route length), (ii) there are no route discovery control packets, only data and acknowledgements, (iii) the communication reliability information is kept locally at each node in the network, not at the source, (iv) Castor does not seek to identify and exclude attackers, and (v) it relies on the simplest trust management assumptions (same as those of the SRP-SSP combination).

5.3. Protocol design overview

Castor operates as follows: when the source sends a packet, intermediate nodes forward it until it reaches its destination, which then responds with an acknowledgement that follows the reverse path back to the source. The basic design elements and ideas behind Castor are discussed next in this section.

Learning from failures. Nodes locally keep per-flow-and-next-hop reliability metrics (§5.5.4), which are updated constantly, based on the arriving acknowledgements indicating success and the acknowledgement timeouts indicating failure. These metrics are used to select the most reliable next hop for each incoming packet. If no reliable next hop exists, or if the recorded history is insufficient for the node to make a choice, the packet is locally broadcasted.

Each node decides whether to unicast or broadcast independently (§5.5.3).

Reliability as the primary performance metric. Protocols that minimize the number of hops or the round-trip time, are susceptible to an attacker advertising shorter routes or setting up wormholes or tunnels to attract traffic and then drop passing data packets. To be robust against such attacks, Castor uses reliability as its primary metric. Since the routing is reliability-driven, Castor is able to detect and react to all causes of packet loss, independent of their nature, be it benign or adversarial.

Response time as the secondary performance metric. For performance reasons, Castor keeps routes short by giving preference to the first neighbor responding with an acknowledgment during the route discovery. However, this neighbor can be routed around if it turns out to be unreliable.

Routing and route discovery as a single process. There are only two message types in Castor: the payload-carrying packets and the acknowledgements. When a packet arrives, from the application layer of the source node or a previous hop, the best next hop is selected based on the kept performance history. When the history is insufficient or indicates likely failure then the protocol seamlessly switches to broadcasts, which serve the function of route discovery.

Emergent reliable routes. With every node estimating and acting on local reliability metrics, Castor is able to locally route to avoid unreliable neighbors. This results in fast global convergence to reliable routes and efficient continuous adaptation under mobility.

Local repair. In contrast to some other protocols, adversarial or benign failures do not always cause the costly network-wide floods to search for better routes. Most of the time, a Castor node has another reliable neighbor to switch to when one neighbor fails. It resorts to broadcasting only when facing severe failures. In most cases, a brief cascade of local broadcasts reaches a part of the network with reliable next hops and unicasting resumes.

Secure, isolated routing state. Nodes make routing decisions independently, based only on locally accumulated neighbor reliability metrics. In other words, nodes are oblivious to any network connectivity information beyond the local neighbors. No routing state is ever exchanged between nodes, which removes the problem of securing the information exchange. State locality and minimal control traffic are also key to Castor's scalability (§5.7.4).

Routing state is stored on a per-flow, not per-destination, basis. A cryptographic scheme ensures that only the packets coming from the flow's source and the acknowledgments coming from the flow's destination can influence the routing state for that flow (§5.5.2). Despite relying on simple trust assumptions, these mechanisms provide a strong protection against routing state pollution by the adversary.

Immutable messages. All message fields in data and acknowledgement packets are immutable, i.e., they do not change as the message is forwarded across the network. This removes the problem of preventing field manipulation by on-route adversaries, e.g., ensuring that the hop count in route responses cannot be decremented.

Resource exhaustion countermeasures. To prevent resource exhaustion attacks and unnecessary flooding when destinations are unreachable, Castor uses a flood rate-limiting mechanism at each node (§5.5.5). Neighbors that cause too many packet floods without subsequent acknowledgements are throttled.

5.4. Assumptions

5.4.1. System model

We consider a wireless ad-hoc network composed of static or mobile *nodes*: computing platforms with wireless transceivers that have a limited communication range. Nodes communicate directly over the wireless channel with their *neighbors*. Nodes assist the other nodes with communication across multiple links (hops). Each node has a unique identity, which can be cryptographically validated if needed. Nodes that conform to system protocols are *correct*, and those that deviate from them are *adversarial*.

Cryptography. Castor requires that for each pair of end nodes, a source s and a destination d, that wish to communicate securely across the network, either s and d share a pre-established symmetric key $K_{s,d}$ or s knows the public key K_d of d. Furthermore, we assume d is able to verify the integrity of the messages sent by s. We also assume that any two correct neighboring nodes can establish a shared secret symmetric key, to authenticate their communication. Neighbor-to-neighbor keys can be established and authenticated in a number of ways, depending on the system instantiation, e.g., through key transport or agreement. Authentication can be performed with the help of local channels, passwords, certificates etc. For example, a certified public key can serve as the verifiable unique identity of a node. Moreover, each correct node can authenticate messages it broadcasts at the data link layer, utilizing symmetric-key schemes such as [134].

Neighbor discovery. Neighbors are discovered by a simple mechanism, such as beaconing. We do *not* require a secure neighbor discovery protocol, which would prevent the adversary from convincing two non-neighbor nodes that they are neighbors [135].

5.4.2. Adversary model

The adversary controls a number of adversarial nodes, which can be *internal* or *external*. The internal nodes are equipped with the same cryptographic material as the correct nodes. For example, a compromised but previously correct node can become an internal adversary. A single adversarial node can appear as multiple network nodes, utilizing multiple compromised identities and cryptographic keys.

An adversarial node can arbitrarily deviate from the protocol definition. In particular, it can drop, modify, and replay any message. The adversary is, however, computationally bounded and cannot break cryptographic primitives.

If beneficial to the attacker, an adversarial node can correctly follow the protocol for any period of time. Adversarial nodes can also act in coordination and mount collusion attacks. Moreover, adversarial nodes can communicate across large distances using fast out-of-band communication links (typically used to mount e.g., *wormhole* and *tunnel* attacks) and jam communication.

The objective of the adversary we consider here is *denial of service*: to prevent communication or in other words to prevent messages from being delivered. In the rest of the chapter, unless stated otherwise, we are concerned with the strongest variant of adversaries, internal and colluding. We do not seek to thwart any adversarial behavior that does not result in packet loss. In particular, we do not address the problem of preventing traffic interception, eavesdropping or analysis.

5.4.3. Metrics

We focus exclusively on flows between correct source-destination pairs. The primary performance metric is the *packet delivery rate (PDR)*. More precisely, we are interested in the *network-layer PDR*. We want to capture the raw network performance in the presence of adversaries, without using any packet retransmission schemes, either at network or upper layers. Further, we are interested in the *bandwidth utilization* per delivered packet.

5.5. The protocol

5.5.1. Message specification

Castor uses two types of messages: PKTs and ACKs. The data packet, **PKT**, is a tuple $(s, d, H, b_k, f_k, e_k, M)$: s and d are the source/destination identifiers; H is the *flow identifier (id)*; b_k is the *PKT id*; f_k is the *flow authenticator*, used for verifying that the PKT belongs to flow H; e_k is an *encrypted ACK authenticator*. Finally, M is the payload, which typically includes an additional integrity protection mechanism.

The acknowledgment packet, **ACK**, has only one field a_k, an *ACK authenticator*, which is used for verifying that the corresponding PKT was delivered to the destination.

5.5.2. Cryptographic mechanisms

To ensure the correct flow state updates, the PKT and ACK fields need to satisfy two properties:

- First, an ACK a_k should only be received by an intermediate node if the destination has indeed received the corresponding PKT b_k. More precisely:

  (A1) given a_k and b_k, any intermediate node can verify that the ACK authenticator a_k corresponds to PKT b_k,

  (A2) given H, PKT b_k and e_k, as well as any number of other correct PKTs and ACKs from flow H (i.e., b_j, e_j, f_j, and a_j, st. $j \neq k$), only the destination can recover a_k.

To satisfy this requirement, one can choose a_k to be a random nonce freshly generated by the source, set $b_k = h(a_k)$ (where h is a cryptographic hash function), and let e_k be an encryption of a_k either with the symmetric key K_{sd} shared between sa and d or the public key K_d of the destination. We assume the former for the remainder of this subsection.

- Second, no node except the source of flow H should be able to generate a PKT b_k that would be verified as belonging to H. We emphasize that we *do not* require an intermediate node to verify that all PKTs originate from some particular source – which is impossible as in our setup the source does not pre-share keys with intermediate nodes. Rather, an intermediate node verifies that all PKTs originate from the same, but arbitrary source. A precise statement of this property is:

(B1) given H, b_k and f_k, any intermediate node can verify that PKT id b_k corresponds to flow H,

(B2) given H, and any number of correct PKTs and ACKs from flow H (i.e., b_j, e_j, f_j and a_j for $j \neq k$), only the source that originated H can generate a new b_k, f_k pair that verifies as belonging to H.

There are many cryptographic schemes that can achieve the properties (A1), (A2), (B1) and (B2); we present here an efficient solution based on Merkle hash trees, as well as an alternative public-key scheme. The latter removes the minor inconvenience of flow restarting, as we explain below. The pseudocode of the Castor protocol given in Fig. 5.2, uses the former scheme.

5.5.2.1. Merkle tree scheme

PKT generation. For each flow that the source s wants to send to a destination d, the source pre-generates: (i) a set of random nonces, $a_1, \ldots, a_w$, the ACK authenticators, (ii) a corresponding set of PKT ids $b_k = h(a_k)$, where h is a cryptographic hash function and (iii) a Merkle hash tree with $h(b_1), \ldots, h(b_w)$ as leaves. The root of this tree becomes the flow id H. The pre-generated values are then used when sending the k-th PKT $(s, d, H, b_k, f_k = [x_1, \ldots, x_l], e_k = E_{K_{sd}}(a_k), M)$; a_k is encrypted using the key K_{sd} shared between s and d. The integers $x_1, \ldots, x_l$ form a sequence of siblings of the vertices on the tree path from $h(b_k)$ up to H (Fig. 5.1), necessary to verify that $h(b_k)$ is a leaf of the tree, i.e., belongs to flow H.

PKT verification. To verify that a PKT $(s, d, H, b_k, f_k = [x_1, \ldots, x_l], e_k, M)$ belongs to flow H, an intermediate node checks whether $h(\ldots h(h(h(b_k)||x_1)||x_2)|| \ldots x_l) = H$, i.e., if $h(b_k)$ is a leaf of the Merkle tree with root H. The $h(\ldots h(h(b_k||x_1)||x_2)|| \ldots x_l)$ is a shorthand notation, in practice the order of concatenations depends on the position of b_k in the Merkle tree. If the above check is successful, the PKT is forwarded and b_k is stored. Otherwise, the

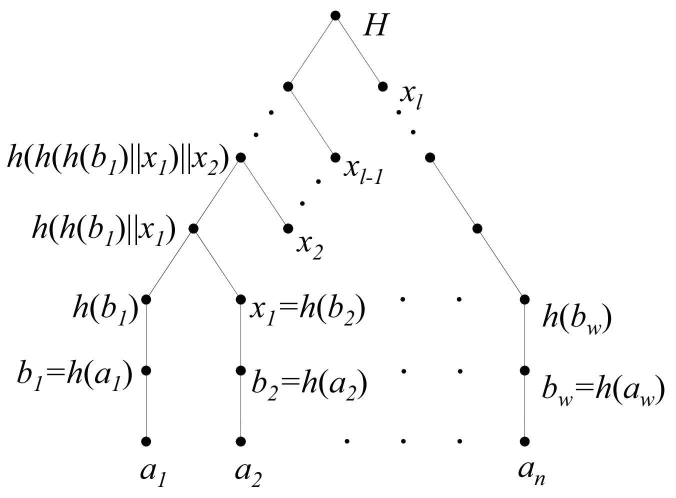


Figure 5.1. The Castor's Merkle tree scheme. The tree construction starts by generating the sequence $a_1, \ldots, a_w$ of random numbers, which are later used to authenticate the ACKs. These numbers are then hashed to generate another sequence $b_k = h(a_k)$. The b_k sequence in turn is hashed to form the leaves of the Merkle tree. The value at each node in the tree computed by hashing the concatenation ($||$) of its children. Assume a PKT containing b_1 and the vector $H, [x_1, \ldots, x_l]$. By checking that $h(\ldots h(h(b_k||x_1)||x_2)|| \ldots x_l) = H$ the receiving node can verify that the PKT belongs to a flow identified by H.

PKT is dropped. Note that unforgeabilty of b_k/f_k follows from the hardness of inverting the hash function h.

PKT verification at destination. In addition to the Merkle tree test, the destination performs additional verification of the PKT. First, it checks whether $b_k = h(D_{K_{sd}}((e_k)))$. Then, it checks the integrity of the payload M. If all tests are successful, d accepts the PKT, and sends the corresponding ACK a_k to the neighbor that delivered the PKT. Otherwise, the PKT is dropped. Note that only the destination is able to generate a correct ACK without breaking the encryption of e_k or finding a pre-image of b_k under h.

ACK verification. Upon receiving and ACK a_k, a node computes $h(a_k)$ and checks whether it corresponds to any stored b_k. If yes, the ACK is accepted and rebroadcasted, and the routing state of the corresponding flow H is appropriately updated. Otherwise, the ACK is ignored.

Flow restarts. When the source exhausts the $a_1, \ldots, a_w$ and $b_1, \ldots, b_w$ sets for a given flow, it has to generate them anew along with the corresponding Merkle tree. This effectively starts a new flow. The in-network state for the old flow can no longer be used for routing and new state needs to be established. In practice, with a high enough value of w, the amortized bandwidth cost of reestablishing the in-network state is negligible, especially when compared to the bandwidth cost of keeping the flow's routes up-to-date under mobility.

The bandwidth overhead also depends on w in another way. The vector $[H, x_1, \ldots, x_l]$ has length $l+1$, where l is the height of the Merkle tree. The height of the Merkle tree is $\lceil \log_2 n \rceil$, thus the per-PKT overhead is logarithmically proportional to n.

In the public-key scheme presented next, the source can generate new a_is and b_is on the fly, hence flow restarts are not an issue.

5.5.2.2. Public key scheme

PKT generation. Assume a flow from the source s to destination d. The source generates a public/private key pair H, H^{-1} used for generating existentially unforgeable digital signatures. The public key becomes the flow id, the private key is kept secret. For the kth PKT the ACK authenticator a_k is a freshly generated random number (a nonce); the PKT id b_k is set to $h(a_k)$ where h is a cryptographic hash function; the flow authenticator $f_k = Sig_{H^{-1}}(b_k)$ is a digital signature of b_k; and $e_k = E_K(a_k)$ is an encryption of a_k either with the symmetric key K_{sd} shared between s and d or the public key K_d of the destination.

PKT verification. To verify that a PKT $(s, d, H, b_k, f_k, e_k, M)$ belongs to flow H, an intermediate node simply verifies that f_k is a digital signature of b_k with key H. If successful, the PKT is forwarded and b_k is stored. Otherwise, the PKT is dropped. The adversary cannot forge a b_k/f_k pair without knowing the secret private key H^{-1}, or breaking the digital signature scheme, both of which are infeasible.

PKT verification at destination. The destination performs additional verification of the PKT. First, it checks whether $b_k = h(K_{sd}\{e_k\})$, using the key K_{sd} shared with the source s. Then, it checks the integrity of the payload M. If all tests are successful, d accepts the PKT, and sends the corresponding ACK a_k to the the neighbor who delivered the PKT. Otherwise, the PKT is dropped. Note that only the destination is able to generate a correct ACK without breaking the encryption of e_k or finding a pre-image of b_k under h, both of infeasible for the adversary.

ACK verification. Upon receiving and ACK a_k, a node computes $h(a_k)$ and checks whether it corresponds to any stored b_k. If yes, the ACK is accepted and rebroadcasted, and the routing state of the corresponding flow H is appropriately updated. Otherwise, the ACK is ignored.

5.5.3. PKT forwarding

Basic forwarding. For every neighbor $j = 1 \ldots n$ and for every encountered flow H, a node i stores a *reliability estimator* $s_{H,j} \in [0, 1]$. Consider what happens when i either 1) receives a PKT, and verifies that it belongs to some flow H (§5.5.2) or 2) i is the source of the PKT. First, i attempts to forward the PKT to the most reliable neighbor, according to the values of all the reliability estimators for the flow H. If no neighbor is deemed reliable, the PKT is broadcasted to all the neighbors in search for more reliable routes. Immediately after the PKT is sent, i starts a timer T_{H,b_k}, which times out after T_{ACK} if the corresponding ACK is not received.

The decision to unicast or broadcast is probabilistic. Let $p_{max} = \max_{j=1\dots n} s_{H,j}$ be the value of the highest reliability estimator for the flow H among all the neighbors $j = 1\dots n$ of node i. The probability that the PKT is broadcasted is $e^{-\gamma p_{max}}$, otherwise it is unicasted to the next hop with the highest reliability estimator. Ties are broken by choosing uniformly at random. The $\gamma > 0$ parameter allows for controlling the bandwidth investment in route discovery depending on the desired packet delivery rates (PDR).

Duplicate PKTs. If a node receives a PKT that it has received before, this PKT is not forwarded again. However, if an ACK corresponding to this PKT was received, the node rebroadcasts the ACK. If an intermediate node receives a PKT with b_k identical to some previously seen PKT, but with a *different* payload or encrypted ACK authenticator e_k, this PKT is forwarded further. This is because an intermediate node cannot tell which of the PKTs with the same authenticator are incorrect or forged. We explain this in more detail in §5.6.2. The T_{H,b_k} timer is not restarted on PKT duplicates.

5.5.4. Updating the reliability estimators

Reliability estimators. The reliability estimator $s_{H,j}$ is an arithmetic average of two reliability estimators $s^a_{H,j}$ and $s^f_{H,j}$. The $s^a_{H,j}$ estimator is updated more frequently than $s^f_{H,j}$ as we explain next. The "a" stands for "all ACKs" and "f" stands for "first ACK".

Both reliability estimators are exponential averages of packet delivery rates. More precisely, let $\alpha^a_{H,j}$ be the running exponential average of successful deliveries and $\beta^a_{H,j}$ the running exponential average of failed deliveries; then $s^a_{H,j} = \frac{\alpha^a_{H,j}}{\alpha^a_{H,j}+\beta^a_{H,j}}$. Upon a failure, the updates are: $\alpha^a_{H,j} \leftarrow \delta\alpha^a_{H,j}$ and $\beta^a_{H,j} \leftarrow \delta\beta^a_{H,j} + 1$. We denote this negative update as $s^a_{H,j}\downarrow$. Upon success, the the reliability estimator is positively ($s^a_{H,j}\uparrow$) updated: $\alpha^a_{H,j} \leftarrow \delta\alpha^a_{H,j} + 1$ and $\beta^a_{H,j} \leftarrow \delta\beta^a_{H,j}$. Initially, $\alpha^a_{H,j} = 0$ and $\beta^a_{H,j} = 1$. Updating of $s^f_{H,j}$ is analogous. The $0 < \delta < 1$ parameter controls how fast Castor adapts; the lower the value the faster the adaptation.

ACK timeout. Consider the case when T_{H,b_k} times out before the corresponding $ACK(a_k)$ is received. The update of the estimators then depends on whether the corresponding PKT was broadcasted or unicasted to some neighbor j. In the former case, no reliability estimators are changed. In the latter case, both $s^a_{H,j}$ and $s^f_{H,j}$ are decreased.

ACK reception. Consider the case when node i receives a valid ACK(a_k) from node j before the T_{H,b_k} timeout. If the corresponding PKT was unicasted and j was the next hop, both estimators $s^a_{H,j}$ and $s^f_{H,j}$ are increased, and the ACK is rebroadcasted, otherwise the ACK is ignored (e.g., when coming from a neighbor that is not j)

If the PKT was broadcasted, the behavior depends on whether it is the first ACK that was received, or not. In the former case, both estimators $s^a_{H,j}$ and $s^f_{H,j}$ are increased, and the ACK is rebroadcasted. Otherwise, only $s^a_{H,j}$ is increased and the ACK is not rebroadcasted.

For both broadcasts and unicasts, only one ACK is accepted per neighbor and per PKT id b_k; the subsequent ACKs are ignored.

Rationale behind the dual reliability estimators. Keeping track of the first ACKs

```
initial values
    $auth_{b_k} \leftarrow$ null, $hop_{b_k} \leftarrow$ null, $acked_{b_k} \leftarrow \emptyset$, $\alpha^{a/f}_{H,j} \leftarrow 0$, $\beta^{a/f}_{H,j} \leftarrow 1$

on receive $PKT(s, d, H, b_k, e_k, f_k = [x_1, \ldots, x_l], M)$ from $j$
    if $h(h(h(b_k||x_1)||x_2)|| \ldots x_l)! = H$ or $T_{H,b_k}.expired$ then return
    if $i == d$ and $auth_{b_k} ==$ null and $M$ not corrupted then
        $a'_k \leftarrow D_{K_{sd}}(e_k)$
        if $b_k == h(a'_k)$ then
            $auth_{b_k} \leftarrow a'_k$
    if $i! = d$ and $auth_{b_k} ==$ null then
        $j_{max} \leftarrow \arg\max_{m=1 \ldots n} s_{H,m}$
        $p_{max} \leftarrow \max_{m=1 \ldots n} s_{H,m}$
        trigger timeout $T_{H,b_k}$ in $T_{ACK}$
        if $rand([0, 1]) < e^{-\gamma p_{max}}$ then
            $hop_{b_k} \leftarrow all$
            broadcast $PKT(s, d, H, b_k, e_k, f_k, M)$
        else
            $hop_{b_k} \leftarrow j_{max}$
            send $PKT(s, d, H, b_k, e_k, f_k, M)$ to $j_{max}$
    if $auth_{b_k}! =$ null then
        send $ACK(auth_{b_k})$ to $j$

on receive $ACK(a_k)$ from $j$
    $b_k \leftarrow h(a_k)$
    if $hop_{b_k} ==$ null or $T_{H,b_k}.expired$ or $j \in acked_{b_k}$ or $(hop_{b_k}! = all$ and $hop_{b_k}! = j)$ then return
    $acked_{b_k} = acked_{b_k} \cup \{j\}$
    if $auth_{b_k} ==$ null then
        $auth_{b_k} \leftarrow a_k$
        broadcast $ACK(a_k)$
        $s^a_{H,j} \uparrow$, $s^f_{H,j} \uparrow$
    else
        $s^a_{H,j} \uparrow$

on $T_{H,b_k}$ timeout
    if $hop_{b_k}! = all$ then
        $s^a_{H,hop_{b_k}} \downarrow$, $s^f_{H,hop_{b_k}} \downarrow$
```

Figure 5.2. Protocol outline. Castor at node i with neighbors $j = 1 \ldots n$. We have excluded the PKT duplicate checking mechanism and flood rate-limiting for brevity. The T_{H,b_k} is the timeout timer fired when waiting for the ACK. The duration of the timeout is T_{ACK}, after which $T_{H,b_k}.expired$ becomes true; $auth_{b_k}$ stores the ACK for PKT b_k, if the ACK was received; hop_{b_k} stores the neighbor to which PKT b_k was sent; in particular **null** if PKT b_k was not received; $acked_{b_k}$ stores the neighbors that sent an ACK for PKT b_k.

with $s^f_{H,j}$ is a performance optimization. The primary routing metric in Castor is the packet delivery reliability, but $s^f_{H,j}$ gives preference to lower round-trip routes, which are typically shorter and consume less bandwidth. A similar method has been used with success in ARAN [125] and SRP [120]. On the other hand, using the second estimator, $s^a_{H,j}$, to keep track of all ACKs allows Castor to obtain more routing information with one broadcasted PKT, leading to faster convergence under attacks and when exploring routes.

5.5.5. Flood rate-limiting

To protect the system from Denial-of-Service attacks exploiting the PKT flooding (§5.6) Castor uses a PKT broadcast rate-limiting mechanism. The mechanism takes advantage of the fact that messages are neighbor-to-neighbor authenticated. For each neighbor $j = 1 \ldots k$ the current allowed broadcast rate r_j is kept. The rates are initially set to one per second. When the broadcast is successful (i.e., when the ACK is received) the allowed rate is multiplied $\alpha = 2$, otherwise it is multiplied by $\beta = 0.5$. The rate values are constrained to the $[r_{min} = 1/10s, r_{max} = 100/s]$ range.

The r_j rate limit is enforced by maintaining a size 3 leaky bucket rate-limiter for each neighbor, which operates as follows. A leaky-bucket is a FIFO queue with a fixed maximum size (three, in our case). The packets can arrive at the queue at any rate but the queue transmits the packets at a constant rate r. When the queue size is already at its maximum the packet is dropped, otherwise it is enqueued. The rate limit can be adjusted by changing r.

The rate-limiting mechanism affects only the PKT broadcasts, PKT unicasts and all other messages are unaffected. We have found that this simple approach provides a strong defense against the flooding-based DoS attacks. We confirm that experimentally in §5.7.5.

5.5.6. Protocol state lifecycle

Castor keeps two types of state: per-b_k and per-flow. The per-b_k state is allocated when a PKT with some b_k is received and there is no state for that b_k yet. The b_k state is deallocated after T_{ACK}, the ACK timeout. The per-b_k state is used to keep track of which next hop was chosen for the b_k packets, whether an ACK arrived and what was its a_k authenticator and to track what packets with the same b_k have been sent (for duplicate PKT handling). The per-flow state is allocated the first time a packet from the given flow H is received and deallocated after $T_{collect}$ period of flow inactivity. The per-flow state contains one reliability estimator for each neighbor.

The biggest component of the memory overhead is the per-b_k state. If we assume a 11Mbps medium entirely saturated with 512-byte packets, a 3 second ACK timeout and a (generous) 128 bytes of state per packet then the upper bound on memory usage is 1056 kilobytes. Note that the same upper bound holds irrespective of the number of flows. The memory consumption is only dependent of the number of packets per second the medium can transmit.

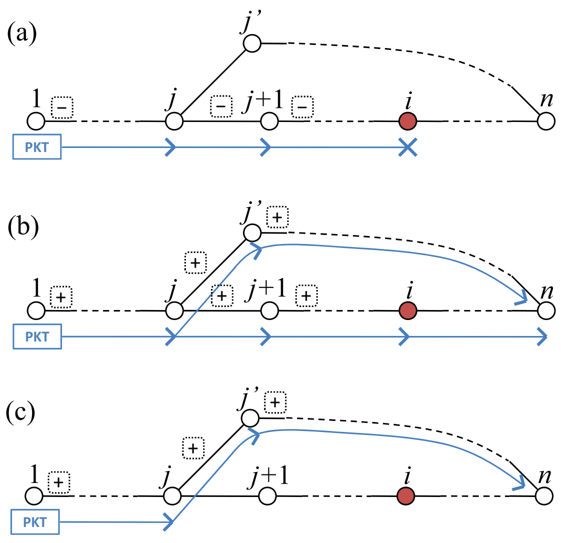


Figure 5.3. Dropping attack defence. Reliability estimator increase indicated by "+", decrease by "-".

5.6. Security analysis

We first show that Castor is resilient to general attacks relevant to any routing protocol, and then do the same for Castor-specific attacks. For presentation clarity, the discussion in this section assumes a static network. In the evaluation section (§5.7) we demonstrate that Castor's security properties also hold under mobility.

5.6.1. General attacks

Packet dropping. An adversarial node drops all (*blackhole attack*) or some (*grayhole attack*) of the packets it is expected to forward. The fraction of packets a grayhole drops can vary over time and dropping can be selective, affecting only specific types of packets.

Consider a packet flow with id H from some correct source 1 to some correct destination n. Assume that the packets are forwarded by nodes $1, 2, \ldots, n-1, n$, and that one of the nodes, say i, is adversarial (Fig. 5.3).

If i drops a PKT, n does not receive it and does not respond with an ACK. Our acknowledgment mechanism guarantees that n is the only node able to generate an ACK for the PKT. On PKT loss (Fig. 5.3(a)), every node $j = 1, \ldots, i-1$ preceding i on the route times out waiting for the ACK and it decreases the reliability estimator $s_{H,j+1}$ for its successor on the route. The more aggressively i drops, the lower the estimators become. One of the following eventually happens: j broadcasts the packet to all of its neighbors (Fig. 5.3(b)) or the reliability estimator $s_{H,j'}$ of some neighbor $j' \neq j+1$ of j exceeds $s_{H,j+1}$, and j forwards subsequent packets to j' (Fig. 5.3(c)). In the case (b), some neighbors of j succeed in delivering the PKT, and respond with a correct ACK, j increases their reliability estimators and new routes are established. Eventually, after another packed drop (a) j re-routes to j', *away* from the source of unreliability. This is the fundamental mechanism through which Castor removes lossy nodes from the routes.

The protocol behavior is similar if the adversarial node i forwards PKTs, but drops ACKs: Nodes $j = 1, \ldots, i-1$ timeout waiting for the ACK and the same mechanism ensures that i is removed from the routes. The only difference to the PKT dropping is that the successors of i on the route receive the ACK and increase their respective reliability estimators. In fact, this is not undesirable, since the resultant routing state updates are correct.

Jamming. In the jamming attack, adversarial nodes prevent communication in their respective ranges. Means can vary: the attack can be mounted on the physical layer or the MAC layer, continuously or intermittently and selectively. Flows ending in the jammed regions are effectively denied communication. For other flows, a jammed region appears as a cluster of black- or gray-hole nodes and Castor is able to route around them.

Wormholes and tunnels. In a tunnel attack, remote adversarial nodes use their fast links to transfer messages out-of-band, appearing to be neighbors. In the more powerful wormhole attack, the out-of-band links are used to almost instantly relay, without any modification, messages received in one location to another remote location in the network. Thus, every node at one end of the wormhole believes to be a neighbor of every node at the other end. Route discovery mechanisms optimizing for hop-count or response time (as Castor does) are attracted by such "shortcuts", and wormholes and tunnels are likely to become a part of many routes. The adversary can take advantage of that and take control over a large fraction of the traffic, which can then be maliciously dropped or corrupted.

Castor uses the tunnels and wormholes opportunistically as long as they allow the traffic to pass through. As soon as the attacker starts dropping the packets, the PKT-ACK loop is broken and Castor turns to alternative, more reliable neighbors for routing, avoiding the lossy tunnels and wormholes. We demonstrate this property experimentally in §5.7.2.

Rushing attack. Even without fast out-of-band links, the adversarial nodes can attempt to place themselves on the routes. In [136], nodes forward broadcasted PKTs as soon as possible by exploiting the MAC layer vulnerabilities. From our perspective, this attack is a weaker variant of a wormhole/tunnel attack: When the rushing nodes start dropping, they will be routed around.

Sybil attack. In a Sybil attack, a single adversarial node appears as multiple nodes to its neighbors, using the cryptographic material of other compromised nodes. As reliability estimators are kept on a per-neighbor basis, routing around a Sybil node is harder: its neighbors have to decrease their reliability estimator for each of the identities of the Sybil node. The Sybil attack is not very different from the wormhole attack. In a sense, the final outcome is identical, a node gains a number of false neighbors. These can potentially become droppers and when they do they are detected and routed around. In our performance evaluation (§5.7), we focus on the most severe attack in this class of neighborhood attacks, the wormhole attack.

5.6.2. Castor-specific attacks

Importance of flow isolation. Castor maintains reliability estimators per-flow, not per-destination, uses flow authenticators to identify the flows and cryptographically binds the PKTs and ACKs. This ensures that 1) in-network state for each of the flows is logically isolated and 2) only the messages originating from the source or the destination can influence the flow's state. Without this, an attacker could generate false PKT-ACK pairs for any chosen flow and maliciously modify the routing state on all the nodes that the false PKTs and ACKs traverse. This could be used, for example, to prevent PKT delivery to a legitimate destination by re-routing the traffic to an adversary-controlled node.

Message corruption and forgery. The adversary can attempt to corrupt any field of ACKs or PKTs. If the payload M of a PKT is modified, the data integrity verification fails at the destination and an ACK is not sent back to the source for that packet. Thus, any node corrupting the payload appears to its neighbors as a packet dropper, and it is routed around.

Replay attacks. As explained in §5.5.2, the adversary cannot successfully forge flow authenticators in PKTs or ACK authenticators. However it can be replay them. Several variants of replay attacks are possible. The objective is to influence state corresponding to legitimate active flows and attempt to reroute the PKTs in order to discard or corrupt them.

First, an adversarial node on a route can replay a forwarded PKT multiple times. Forwarding the PKT to the same neighbor has no effect, as correct nodes forward a given PKT only once. If the PKT is forwarded to different neighbors, all of them try to route it towards the destination. The resulting routing state updates are correct and do not negatively influence the network performance.

Second, the adversary could replay a PKT in another location of the network. As is the case with the local replay attack, such PKTs are routed to the destination, creating correct routing state, again, without any impact on the performance.

The adversary could, however, modify the PKT parts that intermediate nodes cannot recognize as invalid: $E_{K_{sd}}(a_k)$ and M. The Castor nodes forward every distinct copy of the PKT even if the flow authenticators are identical. This ensures the correct copy of the PKT will get through despite the nodes also receiving the malformed clones.

Considering ACK replays, observe that a given ACK can be used to increase an estimator at node j for some neighbor k at most once. Hence, it is not possible to artificially increase an estimator by repeatedly replaying an ACK from one neighbor to another. Each correct node also ignores ACKs that correspond to PKTs it never forwarded. In addition, if a PKT was unicasted to some neighbor j, the corresponding ACK from any node other than j is ignored.

Finally, the adversary can, using its fast communication links, "reenact" a correct flow in some other part of the network. This would require at least two adversarial nodes: one node i

replaying PKTs and another node i' replaying the corresponding ACKs. This creates incorrect routing state all along the reenacted flow. But it has no effect on PDR, as long as the original flow is disjoint from the reenacted flow. If, for some reason, the original flow reaches one of the nodes that are reenacting the flow, then the incorrect routing state delays PKT delivery by forwarding towards i'. This state is then quickly corrected if those misrouted PKTs are not delivered. Overall, this attack cannot be sustained without delivering the PKTs to the correct destination in order to obtain the fresh valid ACKs that would be needed to reenact the flow. However, the delivery to the correct destination defeats the purpose of the attack.

Flooding attack. In Castor, the nodes broadcast the PKTs whenever no reliable route is known. An adversary could use the PKT rebroadcasts as an attack amplification device. A high-rate stream of PKTs could be injected, each PKT belonging to a distinct new dummy flow. This would trigger a network-wide flood for each PKT potentially causing global bandwidth starvation. All the routing protocols relying on flooding for route discovery have this vulnerability: an attacker can trigger floods at a high rate and cause Denial-of-Service. Very few of the existing protocols address that issue. Castor uses flood rate-limiting (§5.5.5) that limits how often a given neighbor can cause a PKT broadcast. We show experimentally (§5.7.5) that this simple measure gives a high level of protection from the flooding attacks.

In Castor, a neighbor is allowed a higher rate when the broadcasts it causes are followed by ACKs. An attacker could exploit that and set up a colluding node in the network that would be ACKing the bogus PKT stream and thus allowing a higher attack rate. However, this also means that there would be enough bandwidth left for some of the benign messages, which weakens the attack. In our experiments we found this attack variant is much more difficult to perpetrate and that it has lower impact on performance than the simple non-ACKed PKT flooding. The rate limiters could be precisely tuned to provide a very strong defence against the ACKed flooding, but we leave this for future work.

5.7. Performance evaluation

Our evaluation is carried out using the ProtoPeer (§A) message passing framework, with JiST/SWANS (`http://jist.ece.cornell.edu/`) employed for MANET modeling. Apart from Castor, we have implemented three other routing protocols: Sprout, SRP and SEAD. We use the default-setting implementation of AODV from the JiST/SWANS library. The choice of protocols covers all combinations of on-demand/proactive and distance-vector/link-state categories (Fig. 5.4). Our Sprout implementation uses the parameters recommended in [130], with one exception: To handle mobility, the maximum number of routes stored at a given time is set to 50. When exceeded, the lowest-ranked route is removed. The SSP [119] protocol is layered on top of SRP [120].

We use the random waypoint mobility model. Nodes send neighbor discovery beacons every $250ms$. A node is removed from the neighbor set if not heard from for $1s$. Unless otherwise stated, the measurements are averaged over 50 independent runs, 1 hour of simulated time each. Every data point in the time-involving measurements is a 10 second average. Each of the 50 runs has distinct node trajectories. The same trajectories are repeated for all the protocols. We show 90% confidence intervals. The experimental setup parameters are summarized in Table 5.1.

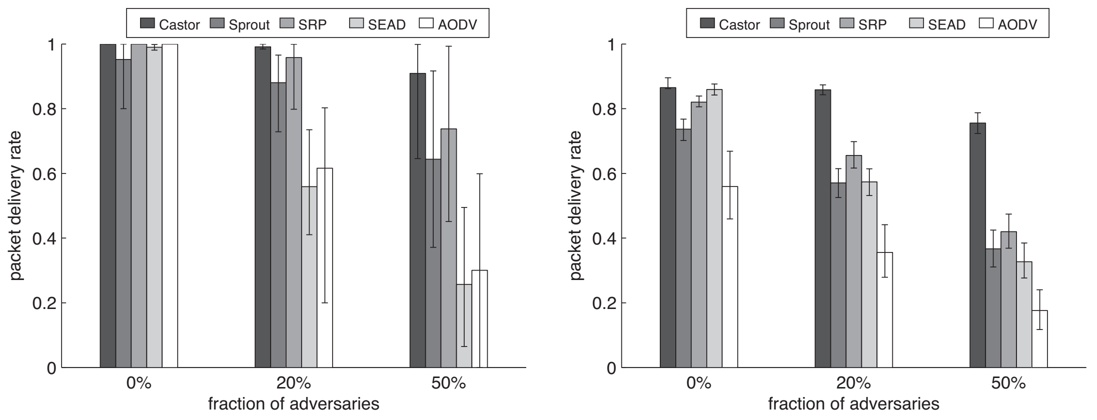
General			
plane size		3km by 3km	
nodes		100, placed uniformly at random, 10 radio neighbors on average	
MAC		802.11b at 1Mbps	
number of flows		5, source-destination disjoint	
flow rate		constant bit rate, 4 packets/s	
packet payload size		256 bytes	
Random waypoint mobility		**Castor**	
min. speed	1 m/s	γ	8
max. speed	20 m/s	δ	0.8
pause time	0s	T_{ACK}	500ms
SEAD			
periodic route update interval	5s		
SRP		**Sprout**	
α, β, δ	0.5	γ	1.25
r_s^{thr}	0	α_{pdr}	0.9
r_s^{max}	1	α_{rtt}	0.9

Table 5.1. Experimental setup

5.7.1. Dropping attack

We implement selective blackholes, dropping all data packets, but allowing control packets (route discovery) through. For Castor, the attacker drops unicasted PKTs, and forwards broadcasted PKT to attract more routes. Figure 5.5 shows the achieved packet delivery rate (PDR), varying the fraction of adversarial nodes. Recall that we look at the network-layer PDR, i.e., there are no retransmissions to mask the packet loss.

The AODV and SEAD protocols do not make any end-to-end checks for packet loss and they are unaware of the blackholes. The performance of the two protocols is thus significantly affected. Sprout and SRP monitor route reliability, and thus significantly improve over SEAD and AODV. However, Sprout and SRP do per-route performance accounting; when the fraction



(a) No mobility (b) With mobility

Figure 5.5. Blackhole attack resilience. We vary the fraction of blackholes in the system with and without mobility. The error bars indicate 90% confidence intervals. Both SEAD and AODV do not have any defenses against the blackhole attacks and suffer the biggest performance drop. Sprout and SRP track the reliability on the per-route basis and when there are more blackholes, most of the routes are likely to contain at least one adversarial node and the two protocols struggle to find an adversary-free path. Castor tracks the reliability per-link and is able to locate and route around the adversarial nodes more accurately.

of attackers is high, most of the routes contain at least one adversarial node and both protocols take longer time to converge on a blackhole-free route. In contrast, Castor keeps reliability records with higher granularity, per-link instead of per-route, and can detect and route around the attackers much faster.

■ BFP	proactive	on-demand
distance-vector	□ SEAD	□ AODV
link-state	■ Sprout	▨ SRP

Figure 5.4. Legend. The suite of evaluated protocols covers the whole matrix of protocol types. Castor could be classified as an on-demand protocol, but does not fall into either the distance-vector or link-state categories, since in Castor no network topology information is ever exchanged.

Failure recovery time. The benefits of storing reliability information per-link rather than per-route are clearly demonstrated by the experiment on Fig. 5.6. Castor recovers in under 30s, much faster than SRP or Sprout. This even more clearly shows that storing reliability information per-link is superior to storing it per-route. This enables Castor to more rapidly pinpoint the location of the blackholes and reach full recovery in under 30s. In contrast, the other protocols converge slower since only the sources are engaged in evaluating the route reliability and not the whole network as in Castor. The number of possible routes to test for loss in Sprout and SRP is combinatorially larger than the number of links to test in Castor.

Even after 30 minutes, Sprout and SRP fail to find reliable routes for some of the flows,

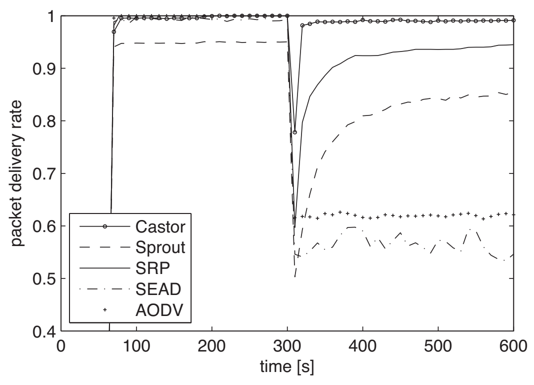


Figure 5.6. Failure recovery time under blackhole attack. There are 100 immobile nodes. The flows start at the 1 minute mark. At the 5 minute mark 20 nodes become blackholes. The results are averaged over 50 independent runs, 5 simultaneous flows in each run. By keeping track of reliability per-link, Castor quickly locates the blackholes and recovers within 30s. This is in contrast to SRP and Sprout, which keep reliability records per-route and converge slower or might entirely fail to find blackhole-free routes.

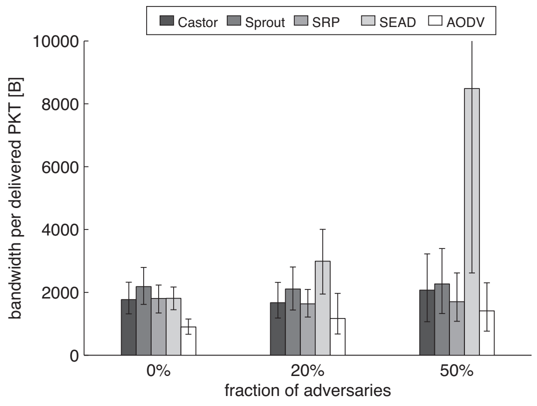
which is reflected in the 50-run averages. AODV and SEAD, not surprisingly, do not recover from the attack.

Mobility. While mobility degrades the performance of all protocols, Castor is the least affected (Fig. 5.5(b)): Nodes observe the topology changes locally and most of the time they are able to select an alternative next hop on the spot or perform a local flood to repair the route. In contrast, the other protocols must either: (i) wait for the new topology information to propagate through the network (Sprout and SEAD) or (ii) wait for the on-demand route (re)discovery to finish (AODV, SRP). In addition, the newly discovered routes must be "evaluated" (Sprout, SRP) for the presence of the black holes.

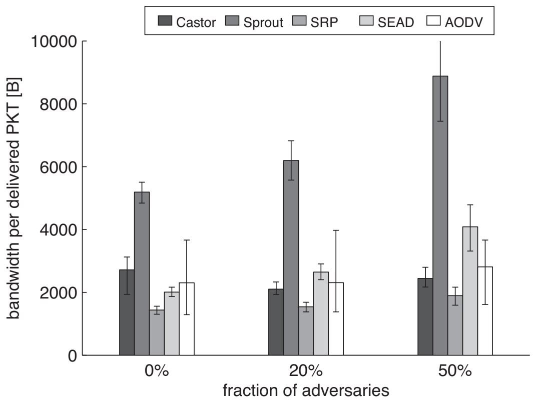
Bandwidth utilization. Bandwidth utilization for the performed experiments is shown in Fig. 5.7. Castor is unique among the five protocols merging the routing and route discovery phases into one. This comes at the cost of including the full 256 byte data payload in the flooded PKTs. However, often Castor responds to changing network conditions by simple re-routing or limited flooding (§5.5.3), which results in amortized bandwidth cost comparable to the other protocols.

The two proactive protocols, Sprout and SEAD, require additional bandwidth for propagating the network topology information. Under mobility, even in the benign case, Sprout uses a substantial amount of bandwidth for link-state updates.

Delay. In the previous experiments we have also measured the end-to-end delay from the moment the packet is sent at the source till the moment it is delivered at the destination (Fig. 5.8). With the exception of AODV, the protocols perform similarly. The AODV implementation that we have used buffers the packets at the source until the route to the desti-

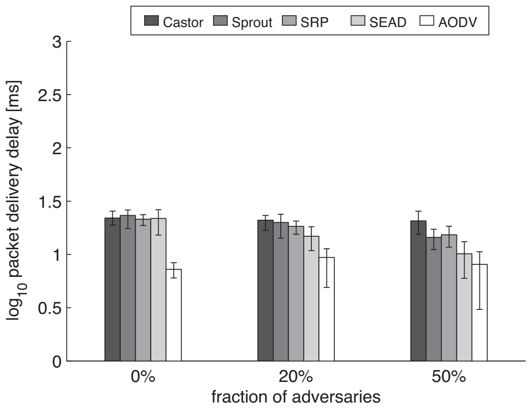


(a)

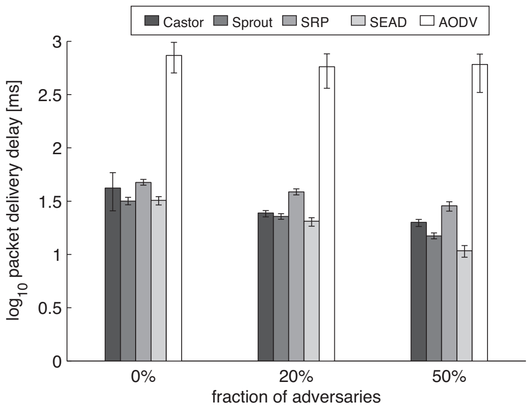


(b)

Figure 5.7. Bandwidth utilization under the blackhole attack. Left: no mobility. Right: mobility. The experimental setup identical to Fig. 5.5. Even though Castor floods the network with PKTs that include the full data payload, the amortized bandwidth cost is comparable to the other protocols. The proactive protocols consume additional bandwidth while exchanging the network topology updates.



(a) No mobility

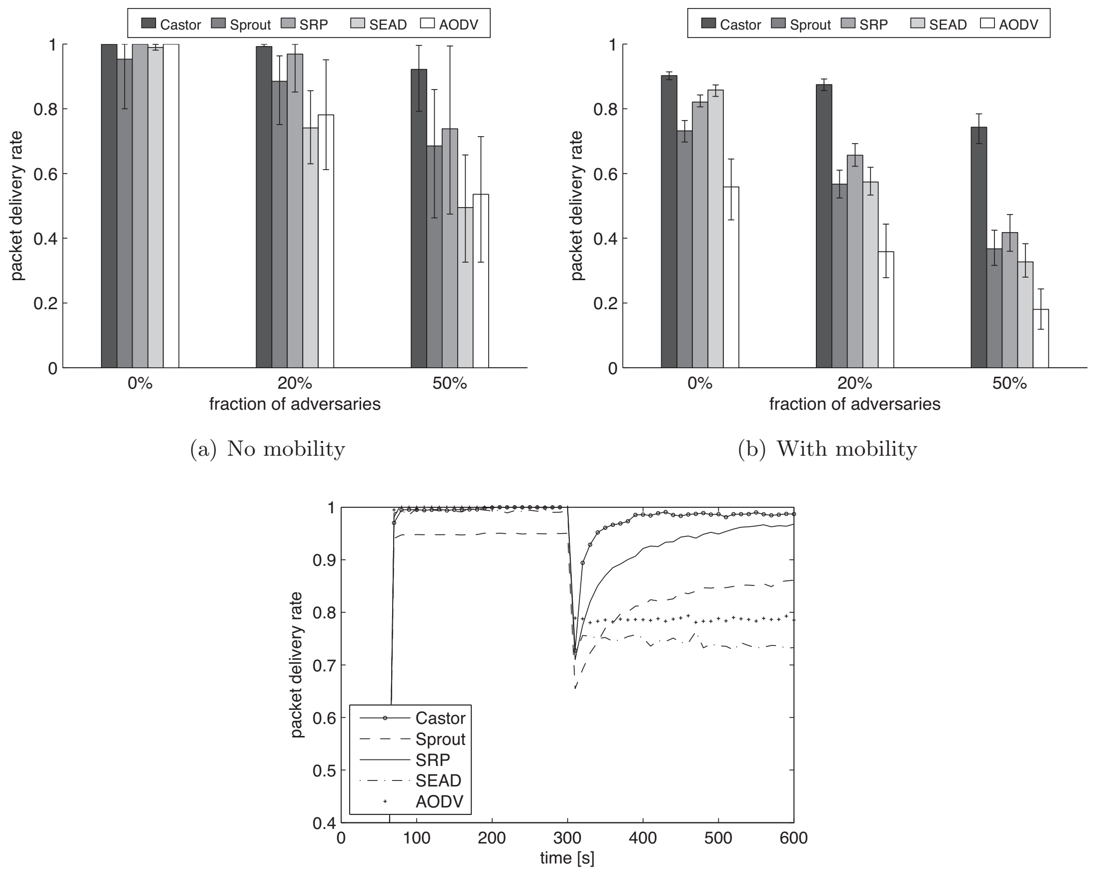


(b) Mobility

Figure 5.8. Packet delivery delay. The experimental setup is identical to Fig. 5.5. We measure the end-to-end delay experienced by the payload packets. Castor performs equally well as the other protocols. In the case of AODV packets are buffered during the frequent route discovery in a mobile network and the time spent in the buffers is the main contributor to the delay.

nation is discovered. The buffer waiting time is the main contributor to AODVs delay. Castor includes the complete payload in the flooded PKTs and does not suffer the delay penalty of the on-demand protocols.

While our simulator models the MAC layer and the time spent on radio transmissions accurately, we do not have accurate models for the per-packet processing time at the nodes. We simply assume packet handling takes a random 5-10ms interval per-packet. This makes our



(a) No mobility

(b) With mobility

(c) No mobility, failure recovery time

Figure 5.9. Grayhole attack resilience. We vary the fraction of grayholes in the system with and without mobility. All protocols except AODV and SEAD defend against the grayholes just as well as the blackholes (Fig. 5.5). However, the time required to detect the attacker is longer.

delay measurements only indicative and precise down to the order of magnitude.

Grayhole attacks. We have so far evaluated the protocols under the black hole attack, where the attacker drops all the data packets. To make the attack more challenging to detect, the attacker could mount a *grayhole attack* in which only a fraction of the data packets are affected. In the next experiment the settings is identical to the blackhole setup, except that now the attacker drops 50% of the packets uniformly at random. The results are presented on Fig. 5.9.

The packet delivery rate of Castor, Sprout and SRP are almost the same as in the blackhole case (Fig. 5.5). However, the protocols take more time to identify the attacker nodes and route around them, when compared to Fig. 5.6. Unsurprisingly, AODV and SEAD suffer a much smaller performance drop, since the attacker affects twice fewer packets.

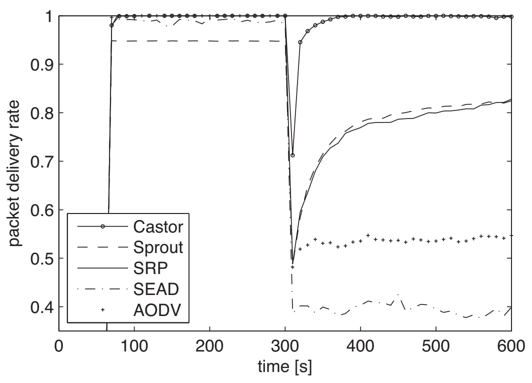


Figure 5.10. Wormhole attack resilience. There are a 100 immobile nodes, a triple exit point wormhole is present. The wormhole is initially passive, but at the 5 minute mark starts dropping all the data traffic. The results are averaged over 50 independent runs, 5 simultaneous flows each. Castor is the only protocol of the five that fully recovers from the wormhole attack.

5.7.2. Wormhole attack

We set up a wormhole with three exit points. The points form an equilateral triangle, each pair of points separated by 1000m. The wormhole is implemented at the radio layer. Initially, the wormhole forwards all the packets. At the 5 minute mark, the wormhole stops retransmitting any data traffic, but still keeps retransmitting the control traffic and broadcasted PKTs. Out of all the wormhole behaviors the we tried, this one had the most severe impact on the performance of all the protocols. We measure how the PDR changes in response to the attack (Fig. 5.10).

Initially, all protocols are attracted to the shorter routes the wormhole offers. After the wormhole stops transmitting data traffic, AODV and SEAD continue to route through the wormhole, because the control traffic goes through. As a result, they suffer from significant packet loss, which is not zero only because some source/destination are located close enough not to be attracted to the wormhole routes. SRP and Sprout gradually switch to routes without any wormhole links. However, because only the source evaluates the routes, convergence is much slower than with Castor. In fact, for some flows SRP and Sprout cannot find an adversary-free route within the simulation time, and thus do not fully recover. Note, that Sprout has not been designed to defend against the wormhole attacks [130] and instead the authors recommend to rely on solutions such as TrueLink [137]. We did not simulate TrueLink. Castor recovers from wormholes completely without any additional wormhole defense mechanisms and their overhead.

Other attacks. We did not evaluate tunnel, rushing or Sybil attacks, as from our perspective they are very similar in nature, but weaker than the wormhole attack. We also omit the evaluation of the replay attacks; as show in §5.6.2, they do not pose a significant threat.

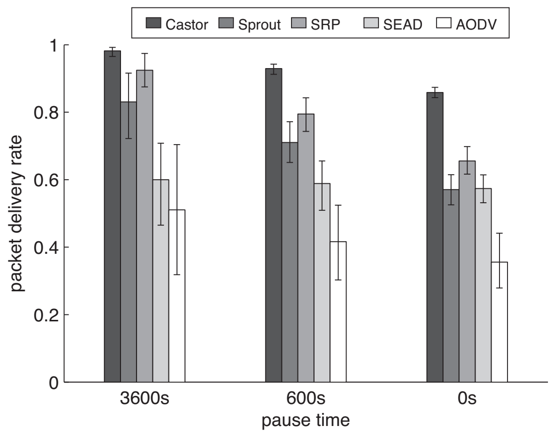


Figure 5.11. Performance under mobility. There are a 100 nodes, out of which 20 are blackholes. We vary the pause time. Castor is fast both at detecting the blackholes and reacting to topology changes, while the other protocols either do not have countermeasures against the blackholes or their detection processes are slower than the rate of change of the network topology.

5.7.3. Performance under mobility

To test how node mobility influences the protocols' ability to detect the adversary, we set the number of blackholes to 20%, and vary the node pause time in the random waypoint model measuring the PDR (Fig. 5.11).

Castor's local failure detection and repair is able to rapidly reroute the PKTs, when nodes go beyond the radio range or the new neighbor turns out to be a black hole. Sprout and SRP need to route more PKTs to determine which routes are reliable and often this process is slower than the rate of change in the topology. AODV and SEAD display constant performance, confirming that its not the mobility that affects them, but rather the attack.

5.7.4. Scalability

We next measure the performance of the protocols for different network sizes, keeping node density constant; 20% of the nodes are blackholes under zero pause time mobility. The results are shown in Fig. 5.12.

The routing paths get longer with the increasing network size and finding an adversary-free path becomes more challenging. The longer paths are also more likely to break due to mobility. Under these conditions Castor still outperforms the other protocols and maintains a 60% packet delivery rate on a 6km by 6km plane with 400 mobile nodes and 80 blackholes.

As the network size increases, the paths get longer and more damage is caused by mobility, making Castor resort to PKT flooding more often, which the bandwidth measurements confirm. The bandwidth utilization of the proactive protocols (SEAD and Sprout) significantly increases. With 400 nodes, Sprout experiences a congestion collapse as the network is overflooded with

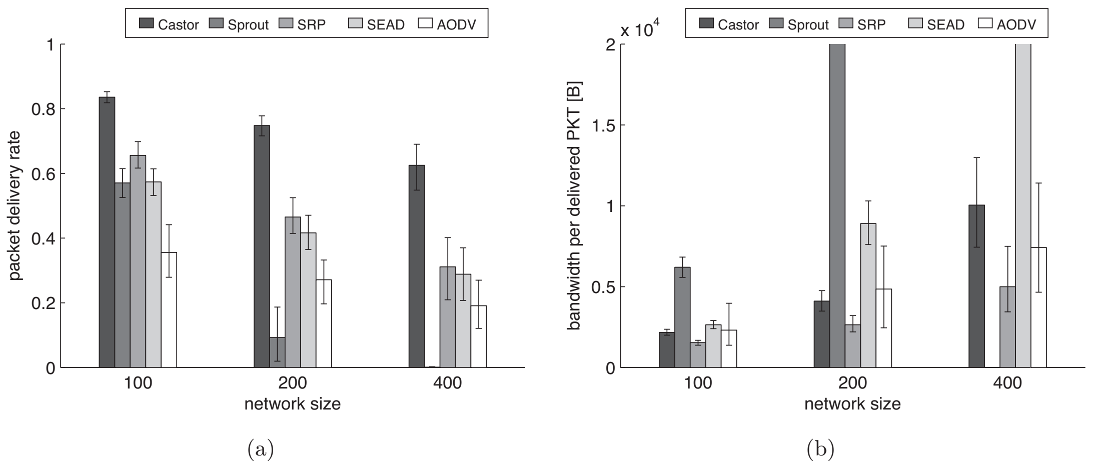


Figure 5.12. Scalability. We increase the number of nodes and the plane size, while maintaining the same node density. There are 20% of blackholes under constant mobility (pause time is zero). The packet delivery rate of all the protocols drops as the network scales up and the longer routing paths break more frequently under mobility. Castor outperforms the other protocols The proactive protocols (SEAD, Sprout) do not scale. With 400 nodes, Sprout suffers from congestion collapse caused by the link-state updates saturating the available bandwidth.

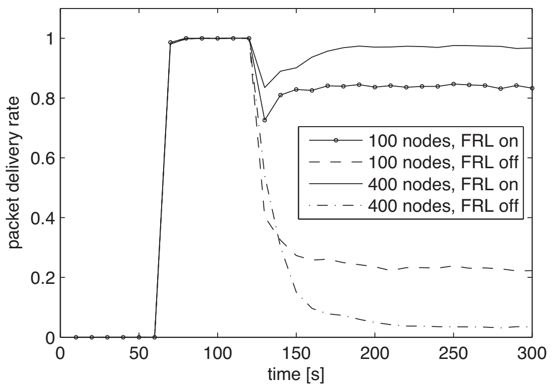


Figure 5.13. Flooding attack resilience. The attack begins at the 120s mark. The five 4pkt/s flows begin at 60s. The curves are averages over 50 independent randomly seeded runs.

link-state updates.

5.7.5. Flooding attack resilience

Without the flood rate-limiting (FRL) mechanism (§5.5.5) Castor is vulnerable to the flooding attack (§5.6.2). In what follows, we experimentally demonstrate the ability of FRL to thwart such an attack.

An attacker controls a single node, which starts 200 new dummy flows per second by broadcasting PKTs, each containing a new unique flow authenticator. With FRL inactive, the attack prevents a large fraction of the traffic from passing through (Fig. 5.13). With FRL active, however, the nodes quickly detect and contain the attacker and reduce the effect of the attack. Complete recovery may not be possible; in some cases the source or destination reside in the neighborhood of the the rapidly broadcasting attacker, which effectively prevents local communication. In a larger network, the routing paths are longer on average and it is easier for the malicious PKT flood to disrupt them. However, when FRL is active in the larger network, the attack is contained to a smaller area relative to the whole plane size, thus the performance decrease is smaller.

5.7.6. Evaluation summary

Resilience to a wide spectrum of attacks. Unlike the other evaluated protocols, Castor does not explicitly separate the route discovery and routing phases and uses the same fault-tolerance mechanism to protect both of them. The cryptographic scheme used in Castor allows each intermediate node receiving an acknowledgment to securely verify that the acknowledgment originated at the destination. Each node locally attempts to improve its routing decisions with a single goal in mind: maximize the number of received correct acks. In this way, Castor's fault tolerance mechanism abstracts away from the causes of the failures, which makes the protocol resilient to a broad spectrum of attacks.

Our evaluation confirmed that Castor is resilient to blackhole and grayhole attacks even when half of the nodes are compromised. Castor is also the only protocol out of the evaluated ones that can fully and consistently recover from the wormhole attacks. As we have argued in §5.6, any attacks that have a local effect and cause loss (e.g., jamming) are eventually routed around by Castor. In addition, because of the autonomy of the nodes and the isolation of the flow state (§5.5.2), Castor is immune to collusion, rushing and replay attacks.

Fast recovery. Sprout and SRP are both source routing protocols and build a reliability model of the network at the sources. The model is constantly updated based on the arriving or timing out acks and the sources adapt to changing loss and delay conditions. In contrast, in Castor the reliability model is distributed; each node locally keeps a performance record of its neighbors. The reliability information is kept exactly where it is needed to make the next hop decisions and can be rapidly updated in reaction to local events. In the other protocols, the information about the changes in the network take longer to reach the route planner (the source), thus substantially slowing down the recovery time compared to Castor. Moreover, Castor keeps the performance records per-link and not per-route, which allows it to pinpoint the failures and the attackers more accurately.

Scalable under mobility. As measured in our evaluation, the proactive protocols (SEAD and Sprout) do not scale. As the network grows, more link-state and routing table information must be exchanged; soon this leaves very little room in the medium for the data traffic, especially under moderately high mobility. In contrast, Castor is an on-demand protocol, which already lowers the bandwidth consumption, but what is more, Castor floods only conditionally and the floods tend to be limited to the part of the network that has been damaged by mobility. As we have shown, this allows the protocol to scale even under a zero pause time mobility and significant (20%) presence of the adversary.

5.8. Discussion

Potential for cross-layer design. In our evaluation we measured the raw packet delivery rate of Castor, without considering packet retransmissions to mask the failures. It is an open question how retransmissions would interact with Castor's fault-tolerance mechanisms and if this would not make the protocol more vulnerable to attacks. Moreover, the protocol uses only two types of messages: PKTs and ACKs, this opens up the possibility of cross-layer integration with the flow control protocols such as TCP or ATP [138], which use exactly the same messaging pattern as Castor.

Sybil attack resilience. We have demonstrated the resilience of our protocol to the wormhole attacks. During a wormhole attack a large number of new nodes appear in the neighborhood of other nodes. The communication to these new neighbors might be cut off at any time by the attacker and yet Castor is capable of recovering. We have assumed neighbor-to-neighbor authentication, which prevents the Sybil attack. However, the Sybil attack follows a similar pattern as the wormhole attack: many node identities controlled by the attacker appear in the neighborhood. This suggests that if Castor is resilient to wormholes, the same defence mechanism could also be effective against the Sybil attack, a point worth investigating.

Optimizing the dynamics. The dynamics of route discovery and failure detection are driven in Castor by two constants: γ and δ. The γ parameter controls the process of deciding whether the current best next hop has an acceptable packet delivery rate or whether the PKT should be broadcast in search for better routes. The δ parameter controls the rate of change of the reliability estimators and the protocol's sensitivity to failures. The two parameters have a significant impact on Castor's performance and attack resilience. We tuned these values experimentally but a more formal understanding of Castor's dynamics is needed to determine the optimal operating point.

Improving the adaptivity. The ACK timeout, T_{ACK}, was fixed to 500 ms for all the scenarios in the evaluation. However, an adaptive timeout, based on the current round trip time measurements for the given destination, could potentially improve the performance by allowing the protocol to react to loss as soon as it happens.

5.9. Conclusions

We proposed Castor, a novel secure communication protocol for ad-hoc networks Despite the very simple PKT-ACK messaging, the protocol is more resilient to attacks than any previously proposed secure communication protocols, as demonstrated by the extensive comparative evaluation. Moreover, Castor abstracts away from the causes of the failure; any network event leading to packet loss, benign or adversarial, is detected and the routes continuously adapt to maintain high packet delivery rate.

Similarly to the Forward Feedback Protocol proposed in the previous chapter, the Castor nodes have a considerable autonomy in making routing decisions. Each node locally keeps a performance record of its neighbors. The reliability information is kept exactly where it is needed to make the next hop decisions and can be rapidly updated in response to local events, which as we confirmed in the measurements, allows for fast failure recovery and fast adaptation under mobility. All that is achieved while relying on weak, and thus more practical, trust assumptions.

Castor is an improvement over FFP from the previous chapters in that it makes the routing secure by cryptographically ensuring the authenticity and the integrity of the messages. This happens at a cost. In Castor, the state is kept on a per-flow basis. Such solution would not scale to the sizes of peer-to-peer overlays, as typically each node participates in routing traffic for a large number of source-destination pairs. To scale, FFP assigns destinations to zones and aggregates the routing state per-zone instead of per-destination. In Castor, on the other hand, keeping the in-network per-flow routing state separate is crucial to the protocol's security and aggregation of the state into zones would likely open the protocol to many attacks. This is a stark illustration of the tradeoff between the scalability and the security of distributed system protocols. It is an important open question whether there is a way to scale Castor's security to the Internet-wide peer-to-peer overlays.

Several other interesting open issues remain. Among them: How would Castor interact with a reliable transfer protocol? Can the PKTs and ACKs be taken advantage of for cross-layer design? How to tune the parameters of Castor, notably the ones controlling the broadcast vs. unicast behavior, to achieve the right balance between route exploration vs. exploitation? Could one extend the reliability estimators to measure both loss and delay? Finally, how do the potential solutions to the above problems affect the protocol security?

Publications: [139, 140]

Chapter 6

Predicting Information Flows in On-Line Social Networks

6.1. Introduction

In the previous two chapters, we have presented two robust routing protocols for peer-to-peer overlays and for mobile ad-hoc networks. The core building block of these protocols is the feedback signal sent after each successful information delivery reinforcing the routes that the information took. In this chapter, we explore whether similar signaling mechanisms play a role in the flow of information in the on-line social networks. We focus on Twitter[1], one of the most popular social sites today.

Twitter users post *tweets*, short messages of up to 140 characters containing a variety of content [141, 142], ranging from daily activity updates, discussions, photos, interesting URLs and random thoughts. Each Twitter user chooses which other users to follow and the tweets from the followed users are aggregated in a single reverse-chronologically ordered stream.

The social graph of followers is a unique and very dynamic communication medium and is host to an increasing number of viral phenomena: breaking news propagation, emergency broadcasts, marketing, public relations, campaigning, activism and many more. Modelling and understanding the flow of information in microblogging systems can potentially lead to more effective use of this new communication medium and provide insights into the underlying sociology.

One of the commonly shared pieces of information on Twitter are the URLs. When a user tweets an interesting URL, her followers, in turn, might re-tweet it to let their followers know about it [143]. This is the basic mechanism through which the URLs spread in the follower graph.

Goal. In this chapter, we focus on characterizing and modelling the information cascades formed by the individual URL mentions in the Twitter follower graph. The information cas-

[1] `http://twitter.com`

cades have been studied before in other Web 2.0 systems, such as Flickr [144], blogs [145, 146], Digg [147, 148] and YouTube [147]. Most of the prior work builds information propagation models to either reproduce cascades with statistical properties matching the empirical observations or to predict how far the information will diffuse in the network given its initial spread. We address a different problem: predicting *which users* will tweet *which URLs* given a training set of existing URL mentions.

Motivation. Accurate prediction of URL mentions is an important enabler of a number of possible applications. First, knowing the tweeting probabilities for each user and URL can be used to generate a ranked list of URLs for each user providing a *personalized recommendation* of URLs that the user is likely to find interesting. For users who follow many other users, this method can be used to prevent information overload by *ranking and filtering* the incoming tweets. Second, aggregating the probabilities per URL quantifies the URL's future potential to diffuse in the social network. This could serve as method for early identification of viral URLs. Third, having an accurate propagation model trained for a specific social network can also help *viral marketing campaigns* to select URL injection points that would maximize the spread of the campaign URL in the network. Finally, a model predicting URL diffusion is not only useful when its predictions are correct, but also when the new data does not match them. A sudden outbreak of anomalous activity that is not correctly predicted by the model most definitely signifies an event worthy of attention. This approach could potentially be used for *spam detection.*

Outline & results. In this chapter, we analyze the spread of 15M (million) unique URLs among 2.7M users based on a 300 hour data set aggregated from Twitter. We measure several statistical properties of the data. First, the Twitter follower graph is a small world with a giant connected component and mean shortest path of 3.61. Second, the tweeting frequencies across the different users and across the different URLs are power-law distributed. Third, the information cascades on the social graph tend to be shallow and wide, having an exponentially distributed depth. Fourth, the cascades for each URL are composed of smaller connected components, whose both number per-cascade and size follow power-law distributions. Finally, the diffusion delay between URL tweets in a cascade is log-normally distributed with a median of 50 minutes.

Based on the above empirical observations, we propose a propagation model that simultaneously takes into account several key factors: content popularity, user influence and the rate of propagation. These factors become the unknown parameters of the model. We use the gradient ascent method to find the parameter values that maximize the number of correctly predicted URL mentions in the Twitter test data. The evaluation shows that the model can predict more than half of the individual URL mentions in the test data set while having a less than 15% false positive rate.

6.2. Related work

Diffusion processes in networks have been studied in a variety of different areas. In epidemiology, extensive research has been done in predicting the spread of disease in populations. A number of models draw on that research to model the spread of information in the social graph as viral processes [149, 150, 151]. However, epidemiological models generate bimodal distributions of infection sizes, i.e., either most or few nodes are infected with information. This does not explain the observations from Web 2.0 systems well, which has led to development of alternative models, e.g., in which the transmissibility falls off with distance from the source [146]. Information diffusion has been studied in many Web 2.0 systems, including Flickr [144], blogs [145, 146], Digg [147, 148] and YouTube [147]. This work, in general, focuses on building models that generate information cascades whose statistical properties match the empirical observations; or models that predict how far the information will spread in the network. Unlike the existing work, our aim is to make predictions at a much finer level of granularity per user-URL pair, rather than modelling larger-scale aggregates of user activity.

Several models for diffusion on graphs have been proposed. In the *linear threshold model* [152] each node has an associated threshold value. If the number of infected neighbors of a node exceeds its threshold then the node itself becomes infected. The *independent cascade model* [153] associates a fixed spreading probability per graph edge and allows each node to attempt infecting another node only once. Further studies have generalized these models [154]. In all of the work, the propagation model's parameters are either constant or drawn from distributions. In our approach, we find the optimal parameter values by training the model on actual information diffusion data. Earlier measurement studies of Twitter measured the properties of the follower graph and looked at its changes over time[155, 156], others studied the geographic distribution of users [141] and examined the phenomenon of re-tweeting [143]. Most recently, the results of Kwak et al. [157] look a number of properties of the Twitterspehere. Many of the results align with ours, in particular, retweet trees are measured to be shallow with power-law distributed size.

6.3. Data set

6.3.1. Data acquisition

Tweets. For 300 hours, starting on Thu, 10 Sep 2009 19:56:47 GMT the Twitter Search API was continuously queried for the search string `http`. The text of each tweet returned by the query was parsed for any URLs and user names it contained.

URLs. Each URL mentioned in the tweets was stored. If the URL was created by one of the popular URL shortening services, HTTP redirects were recursively followed to expand the URL to its original form. All the URLs were also URL-decoded to ensure uniform representation under the percent-encoding (`%xx`) notation.

component size	2	3	4	5	6	2483290
#components	513	31	6	2	1	1

Table 6.1. The size distribution of the strongly connected components of the follower graph.

User graph. For each tweet, we queried the Twitter API for the metadata about the tweet's author as well as all the users that the author follows.

The final outcome is a dataset of timestamped URL mentions together with where they happened in the social graph.

6.3.2. Limitations

Stream continuity. The Twitter search API allows for specifying the minimum timestamp for the tweets returned in the query results. In this way, the complete stream of tweets can be systematically downloaded. The combined availability of Twitter and our crawling infrastructure during the data acquisition period was 99.6%, which means that at most 0.5% of the tweets can possibly be missing from the stream.

User graph. The user graph contains only those users whose tweets appeared in the stream, i.e., only users that during the 300 hour observation period posted at least one public tweet containing a URL. The graph does not contain any users who do not mention any URLs in their tweets or users that have chosen to make their Twitter stream private.

For each newly encountered user ID, the list of followed users was only fetched once. Our data set does not capture the changes occurring in the user graph over the observation period.

6.4. Follower graph properties

The user graph consists of 2.7M nodes connected with 218M arcs. Each arc points from the *follower* to the *followee*.

Component sizes. The user graph is directed, the distribution of the sizes of the strongly connected component sizes is in Table 6.1. The Twitter user graph has a giant connected component encompassing the majority of the users. This is characteristic of many other social networks, both on-line and real-world (§6.2).

Degree distribution. We have measured the degree distribution in the user graph (Fig. 6.1). Both the in-degree and the out-degree distributions have tight log-normal fits.

Path lengths. An important structural metric in graphs is their shortest path distribution. The Twitter graph is a small world (Fig. 6.4) with a mean directed path length of 3.61.

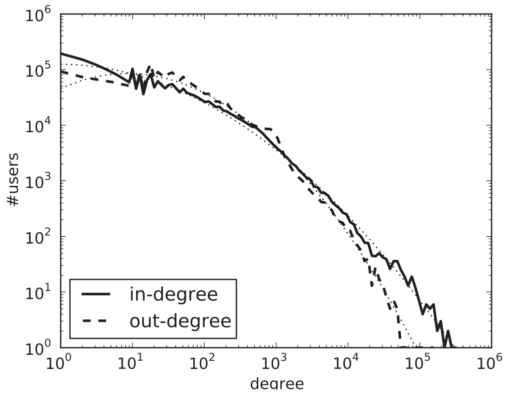


Figure 6.1. User graph degree distribution. In the follower graph, each arc points from the follower to the followee. Both the node in-degree and the out-degree distributions are log-normal (dotted lines, in-degree: $\propto \ln\mathcal{N}(\mu = 6.85, \sigma^2 = 6.55)$, out-degree: $\propto \ln\mathcal{N}(\mu = 5.99, \sigma^2 = 3.84)$))

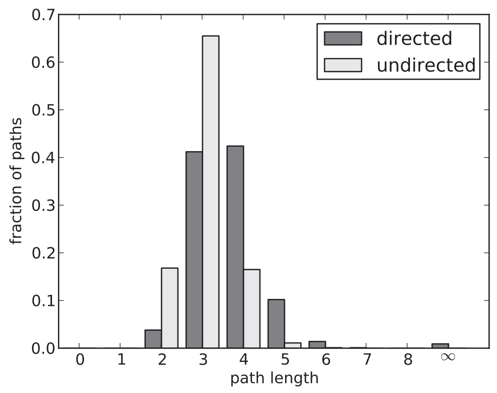


Figure 6.2. Shortest path length distribution. We compute the shortest path for each of the 1000 randomly chosen source-destination pairs. There are two cases: the paths are allowed to traverse along the arcs only (*directed*) or in both ways ignoring the direction of the arc (*undirected*). The ∞ symbol indicates that no path exists for a source-destination pair.

6.5. URL sharing activity

URL mentions. Users can include more than one URL as part of the tweets. In our set of 27M tweets there are 15M unique URLs mentioned. Each tweet has a maximum length of 140 characters, which puts a limit on the maximum number of URLs that can be included in a single tweet. The vast majority of the tweets contain no more than two URLs.

User activity. Users vary wildly in how frequently they tweet URLs. For the two different metrics we have used, the user activity is power-law distributed (Fig. 6.3).

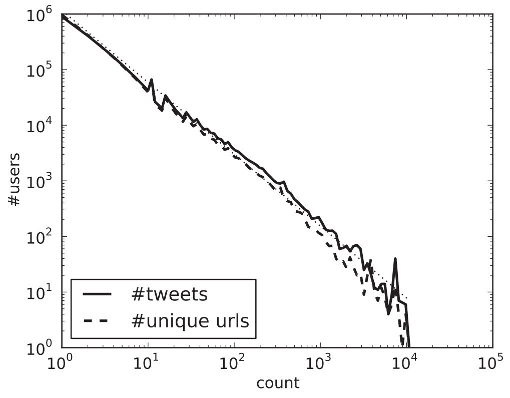


Figure 6.3. User activity. The user activity measured by how many tweets or unique URLs the user posted. For both metrics, the user activity is power-law distributed (dotted line: $\propto x^{-1.25}$).

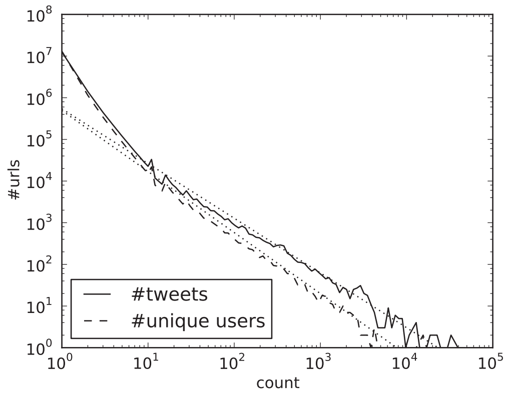


Figure 6.4. URL popularity. URL popularity measured by the number of tweets and number of unique users that mentioned the URL. In both cases we obtain the power-law distributions, $\propto x^{-1.3}$ for tweets and $\propto x^{-1.45}$ for unique users.

URL popularity. The different URLs are mentioned with different frequencies in the Twitter social network, again fitting the power-law distribution. (Fig. 6.4).

6.6. Information cascades

When `@alice`, a Twitter user, mentions a URL, all of her followers can immediately see her tweet in their feeds. When `@bob`, a follower of `@alice`, sees a URL coming from her, he can also tweet that URL to let his followers know about it. This is one of the common mechanisms through which URLs propagate in the Twitter social network.

We have chosen the URLs as the subject of the study since they are a low-noise language-independent signal whose propagation can be easily tracked using the Twitter API.

Retweets. Often when users tweet a URL that was found in another user's feed, they give credit to the original URL poster. This phenomenon became known as retweeting [143]. If `@bob` wants to give credit to `@alice`, he prepends her message with `RT @alice:` followed by the text of the original tweet of `@alice`. `RT` stands for re-tweet. We only focus on the tweets containing URLs, even though any tweet can be retweeted in this way.

Retweets are a strong indication of the direction of information flow in the Twitter social graph in that they explicitly identify the source.

6.6.1. Definitions

We next define two types of information cascades: *F-cascades*, for which the flow of URLs is constrained to the follower graph and *RT-cascades*, for which we disregard the follower graph and use the who-credits-whom data from the retweets.

Let $G(V, E)$ be the *Twitter follower graph* consisting of users from the set V. Graph G is directed, there is an arc $(v_1, v_2) \in E$ in the graph iff the user v_1 follows user v_2. Let U be the set of all URLs.

Let a ***F-cascade*** $F(u)$ be a graph of all the users that have tweeted the URL $u \in U$. An arc (v_1, v_2) exists in $F(u)$ iff: 1) v_1 and v_2 tweeted about u, 2) v_1 mentioned u before v_2 and 3) v_2 is the follower of v_1.

Similarly, the ***RT-cascade*** $R(u)$ is a graph of all the users that have either retweeted the URL $u \in U$ or have been credited as the source of the URL in a retweet. An arc (v_1, v_2) exists in $R(u)$ iff: 1) v_1 tweeted about u, 2) v_1 mentioned u before v_2 and 3) v_2 credited v_1 as the source of the URL u.

Each retweet credits only a single user[2], which makes each RT-cascade a forest consisting of trees. On the other hand, in the F-cascades, each node can have several followees that mentioned a URL before it and in general F-cascades are directed acyclic graphs. Both F- and RT-cascades are not necessarily connected graphs. Each weakly connected component of an F- or RT-cascade is a *subcascade*. The *root* of a subcascade is the subcascade node that mentioned the URL first.

6.6.2. Cascade properties

RT-cascades vs. F-cascades. Even though there is a large overlap between the two types of cascades, 33% of the retweets that we have observed credit users that the retweeters do not follow.

Number of subcascades. Each cascade consists of one or more subcascades. The number of subcascades per cascade is power-law distributed (Fig. 6.6).

Subcascade size. For each cascade the subcascades not only vary in number, but also in size. The distribution of the subcascade sizes taken across all the cascades is in Fig. 6.6. Again,

[2] we parse out only the user name that immediately follows the `RT` characters

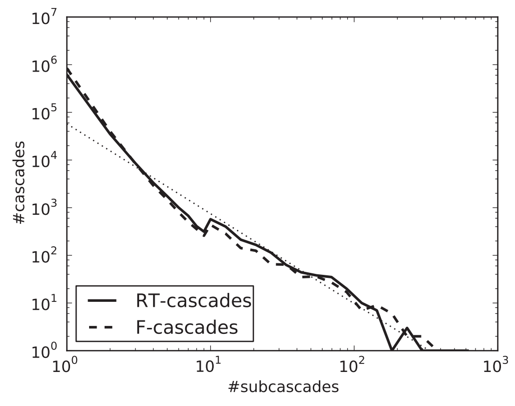


Figure 6.5. Number of subcascades per cascade. For each cascade we look at how many subcascades it consists of and plot the distribution. There is a clear power-law falloff (dotted line: $\propto x^{-1.83}$).

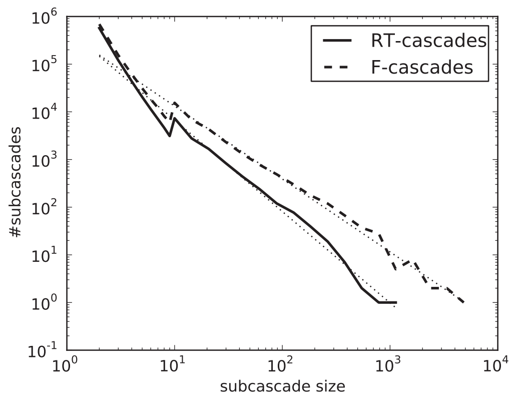


Figure 6.6. Subcascade sizes. The distribution of subcascades sizes taken across the set of all the subcascades. Power-law fits (dotted lines): $\propto x^{-1.9}$ for RT-cascades and $\propto x^{-1.5}$ for the F-cascades.

power-law fits describe the data well.

Subcascades are shallow. For a given node i and URL u, the distance from the subcascade root to i is an important metric characterizing the information flow in the network. Taken across all the subcascades, the maximum distance to root falls off exponentially (Fig. 6.7). The average distance falls off even faster.

The distances to the root are short, even when compared with the already short average path length in the follower graph (Fig. 6.4). One hypothesis explaining this data could be that when a user receives some interesting URL along an path longer than 1, then that user is very likely to start following the original source of the URL (subcascade root), thus shortening the potential future paths to one hop. Twitter does not place any constraints on the number of followers or followees, which allows the participants to optimize the information pathways for

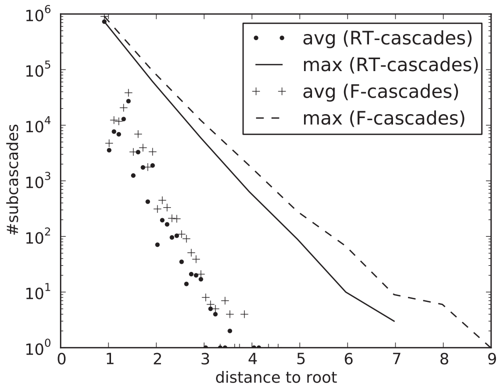


Figure 6.7. Subcascades are shallow. For each subcascade we measure the average and maximum distance from the root to each of the subcascade nodes. We plot the distributions of these values taken across all the subcascades. Both distributions fall off exponentially.

efficiency. Verifying this hypothesis would require additional data on how the follower graph evolves over time. We use the shallowness property of the cascades to reduce the computational effort of training our propagation model (§6.8), by considering only the one hop neighborhoods of nodes that have already tweeted a given URL.

The influence of the powerusers. A number of followers per user varies over several orders of magnitude (Fig. 6.1). The URLs tweeted by the highly connected users reach large audiences and are likely to be (re)tweeted by their followers. Figure 6.8 shows that indeed this is the case. However, the causality is likely to be bidirectional: the users's URLs are tweeted more because they have many folllowers, but also they have accumulated many followers because what they tweet tends to be interesting and viral.

Rate of diffusion. When `@alice` tweets a URL some time elapses until her followers see that and tweet the URL. The *diffusion delay* for URL u and user i is the time from the moment the first of the followees of i tweeted u until the moment i tweeted u. The diffusion delay taken across all the (u, i) pairs is log-normally distributed with a median of 50 minutes (Fig. 6.9). The log-normal diffusion delay is incorporated into our propagation model (§6.7) to improve its accuracy.

6.7. Propagation model

Given the observations made in the previous sections, we construct two models of information propagation in social networks (§6.7.1). The models take into account the influence of users on one another, the virality of the URL and the diffusion delay. These factors have corresponding unknown parameters in the model. The optimal parameter values are found using the gradient

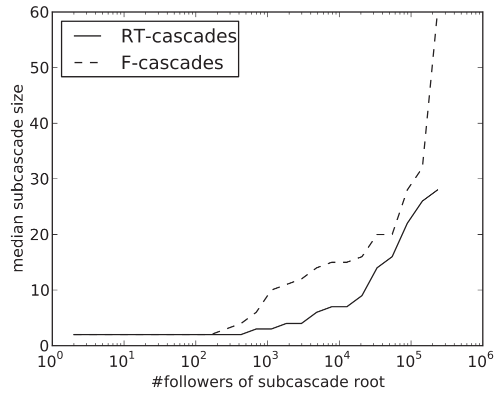


Figure 6.8. The influence of the powerusers. For all subcascades we plot the subcascade size against the number of followers of the subcascade root. For users that have more than approx. 500 followers there is a larger than 50% chance that if they are the root then their URL will be (re)tweeted more than once (i.e., median>1).

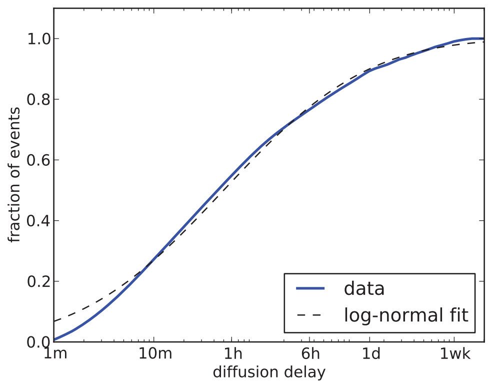


Figure 6.9. Diffusion delay. Log-normal distribution of the delay between the user first hears about a URL from one of the followees until the moment the user tweets about that URL. The fit is $\ln\mathcal{N}(\mu = 3.91, \sigma^2 = 6.86)$. The median is 50 minutes. All URL mentions, not only retweets are taken into account.

ascent training algorithm (§6.7.2) with the goal of maximizing the number of predicted URL mentions while at the same time minimizing the number of false positives.

6.7.1. Tweeting probability

At-Least-One (ALO) model. Let the probability that the user i retweets URL u be:

$$p_i^u = A(\alpha_{ji}, \beta_i, \gamma_u)T(\mu_i, \sigma_i^2, t_i^u) \tag{6.1}$$

The A component represents the time-independent (atemporal) part and T the time-dependent (temporal) part.

The A component is defined as:

$$A(\alpha_{ji}, \beta_i, \gamma_u) = 1 - (1 - \gamma_u \beta_i) \prod_{j:i \to j} (1 - \gamma_u \alpha_{ji} p_j^u) \tag{6.2}$$

The parameters are as follows: $\gamma_u \in [0, 1]$ is the virality of URL u, $\beta_i \in [0, 1]$ is the baseline probability of user i tweeting any URL and the $\alpha_{ji} \in [0, 1]$ parameters define the influence of a followee j over user i. The notation $i \to j$ means "i follows j".

The expression for A is the probability that at least one of the events happens: 1) the URL u is viral (γ_u) and the user is influenced by some followee j (α_{ji}) that tweeted u with probability p_j^u or 2) u is viral and the user decides to tweet it by herself or is influenced by some unknown entity, as quantified by β_i.

The expression for T follows our observations from §6.6.2 that the empirically observed diffusion time t_i^u of user i tweeting about u is log-normally distributed. The cumulative distribution function is parametrized by μ_i and σ_i:

$$T(\mu_i, \sigma_i^2, t_i^u) = \frac{1}{2} erfc(-\frac{\ln t_i^u - \mu_i}{\sigma_i \sqrt{2}}), \tag{6.3}$$

where *erfc* is the complementary error function.

Linear Threshold (LT) model. The ALO model assumes that it is enough to be influenced by one user to cause that user to tweet. The linear threshold (LT) model [152] generalizes over that by introducing a per-node threshold which must be exceeded by the cumulative influence from all the followees for the user to tweet. We replace the A component by:

$$A(\alpha_{ji}, \beta_i, \gamma_u) = s(\gamma_u(\beta_i + \sum_{j:i \to j} \alpha_{ji} p_j^u)), \tag{6.4}$$

where:

$$s(x) = \frac{1}{1 + e^{-a(b-x)}} \tag{6.5}$$

is the sigmoid serving as a continuous thresholding function.

In our experiments, although the threshold is fixed, the α_{ji} parameters can be adjusted to achieve the effect of having a variable per-node threshold.

6.7.2. Training

The model is parametrized by $\alpha_{ji}, \beta_i, \gamma_u, \mu_i, \sigma_i$. These parameters are unknown and are learned in the following process. The training data set S consists of tuples $(i \in V, u \in U, t_i^u \in [0, \infty], F \in \{true, false\})$, where i is the tweeting user, u is the URL, F is the flag indicating whether u tweeted i and t_i^u is the *exposure time* measured as the longest time since one of the followees of i mentions u until either: 1) i mentions u ($F = true$) or if it does not then 2) t_{end}, the time at the end of the training data set ($F = false$). The positive events ($F = true$) cover all the actual URL mentions. The negative events ($F = false$) are generated as follows. For each positive event $(i, u, t_i^u, true)$ and each follower j of i that has not tweeted u, generate $(j, u, t_j^u, false)$. This reflects our empirical observation that the URL cascades are shallow (§6.6.2). As we show in the evaluation (§6.8), including only one-hop follower neighborhood of the nodes that tweeted u provides sufficient training data for the model, while preserving the computational feasibility.

Optimization goal. Given the training data set S, the goal is to find the optimal set of parameters α_{ji}, β_i, γ_u, μ_i, σ_i. Optimality is defined as follows.

If the (i, u) pair occurs as an event in S, it does so only once either as a positive or negative event. If the event is positive then this defines the *target probability* $p_i^u = 1$ otherwise $p_i^u = 0$.

Our model defines the *estimated probability* $\hat{p_i^u} = A(\alpha_{ji}, \beta_i, \gamma_u)T(\mu_i, \sigma_i^2, t_i^u)$. The goal of the optimization is to bring the estimated probabilities $\hat{p_i^u}$ as close to the target probabilities p_i^u as possible.

The $\hat{p_i^u}$ variables are a function of other $\hat{p_j^u}$ variables. Finding the values of these variables would entail solving a system of non-linear equations, which is an intractable problem for the current $\hat{p_i^u}$ definitions. To overcome this, we compute each $\hat{p_i^u}$ directly based on the known target probabilities p_j^u from the training data set (zeroes or ones).

Accuracy metrics. To measure the accuracy of the $\hat{p_i^u}$ predictions we borrow two concepts from the information retrieval literature: *precision* and *recall*. Let k_{tp} be the number of true positives, k_{tn} true negatives, k_{fp} false positives and k_{fn} false negatives. Then the precision is:

$$p = \frac{k_{tp}}{k_{tp} + k_{fp}} \tag{6.6}$$

and the recall is:

$$r = \frac{k_{tp}}{k_{tp} + k_{fn}} \tag{6.7}$$

Intuitively, precision answers the question: among all the positives predicted by the model, how many of them are true? and for recall: among all the positives, how many of them are predicted by the model?

The k_{tp}, k_{tn}, k_{fp} and k_{fn} numbers are obtained by comparing the true values p_i^u with the predictions of the model $\hat{p_i^u}$. In our case, p_i^u is binary, but $\hat{p_i^u}$ is a value from $[0, 1]$. We approximate the k values by computing their expected values given the estimated probabilities $\hat{p_i^u}$:

$$k_{tp} = \sum_{(u,i) \in S_p} p_i^u \tag{6.8}$$

$$k_{fp} = \sum_{(u,i) \in S_n} p_i^u \tag{6.9}$$

$$k_{fn} = \sum_{(u,i) \in S_p} (1 - p_i^u) \tag{6.10}$$

$$k_{tn} = \sum_{(u,i) \in S_n} (1 - p_i^u) \tag{6.11}$$

The sums iterate over the trainig set S, where S_p is the set of positive events and S_n is the set of negative events. The precision p and recall r are then defined in terms of the k values as in the previous paragraph.

Ideally, we would like both the precision and recall to be high. A commonly used metric that combines the two is the *F-score*[3] defined as the harmonic mean of precision and recall $F = \frac{2pr}{p+r}$, which simplifies to:

$$F = \frac{2k_{tp}}{2k_{tp} + k_{fp} + k_{fn}} \tag{6.12}$$

The goal is to maximize F under the α_{ji}, β_i, γ_u, μ_i, σ_i parameters.

Iterations. We use the gradient ascent method for finding the optimal $F(X)$ under $X = (\alpha_{ji}, \beta_i, \gamma_u, \mu_i, \sigma_i)$. The process is iterative. Starting from the randomly initialized X_0 we obtain successively better approximations by computing $X_{t+1} = X_t + c\nabla F$, where ∇F is the gradient of F and c is the constant controlling the convergence rate. The parameters in X are forced to stay within their bounds in each iteration. After a fixed number of iterations we obtain the optimal value of X.

Computational complexity. Each iteration above computes a new value X_{t+1} as well as the gradient ∇F. Computing X_{t+1} involves 1) iterating through training set S and computing the $\hat{p_i^u}$ values, 2) computing the k values and 3) the F. In step 1) each α_{ji} is referenced at most once in the product and also each event in T is referenced at most once (one $\hat{p_i^u}$ computed per event), hence the computational complexity is $O(\max\{|S|, |X|\})$. In step 2) each event in T is reference at most once and step 3) has a constant cost. The complexity of computing X_{t+1} is $O(\max\{|S|, |X|\})$.

Computing ∇F involves computing the partial differentials of all the variables in X. This can be broken down to computing the differentials of individual terms comprising the sums in the k values. There are $|S|$ such terms. Each term can possibly reference only one γ_u among all us and only one α_{ji},μ_i,σ_i,β_i among all the is, hence computing ∇F involves summing over at most $5|S|$ terms (once for each of the 5 parameter classes α,β,γ,μ,σ). Computing each term involves iterating over the relevant α_{ji}, however each α_{ji} is referenced only once. The complexity of ∇F computation is hence $O(\max\{|S|, |X|\})$.

The iteration progresses until the desired numerical precision is reached. The computational complexity of m iterations is $O(m \max\{|S|, |X|\})$.

[3] http://en.wikipedia.org/wiki/F1_score

6.8. Model evaluation

6.8.1. Setup

Training and test data sets. We divide our 300 hour data set (§6.3) into two 150 hour parts. The *training data set* S is generated based on the events from the first 150h window excluding the URLs that were mentioned less than 5 times. For each URL u mentioned by user i we add a positive entry $(i, u, t_i^u, true)$ to S (§6.7.2). For each URL u not mentioned by user i, if there exists at least one followee j of i that mentioned u we add a negative entry $(i, u, t_i^u, false)$ to S. To reduce the size of the training data set, a negative entry is only added if i has tweeted some URL after j, otherwise we assume j's influence on i is too low to warrant an entry in the training data set. In other words, the models will only be trained to predict tweeting probabilities for the users that are one-hop away in the follower graph from the nodes that have mentioned u and only for arcs along which URL has been transmitted. Predicting mention probabilities for all (u, i) pairs (not only the one-hop neighborhood) is computationally challenging (§6.9).

For positive entries, the delay t_i^u is set to the time that elapsed from the beginning of the first 150h window till the first mention of u by i, but only if no followee of i mentioned u. Otherwise, t_i^u is set to the time elapsed from the earliest mention of u by a followee of i till the first mention of u by i (i.e., the diffusion delay, §6.6.2). For negative entries, the delay t_i^u is set to the time that elapsed from the earliest mention of u by a followee of i until the end of the first 150h window.

For the *test data set* we pick 100 random URLs that were mentioned at least 10 times both in each of the first and the second 150h windows. All the mentions of these URLs in the second 150h window become the test data set. The goal is to predict these events as accurately as possible.

There are approximately 700k positive and 9M negative entries in the training data set with 500k unique users and 50k unique URLs. The test data set has 5.2k URL mentions that need to be predicted.

Models. Our evaluation covers the following models (§6.7.1): **ALO** - the at-least-one model (§6.7.1), **LT** - the linear threshold model, **LTr** - a simplified version of LT that has one α_j parameter per influencing node instead of having an α_{ji} parameter for each pair of influencing and influenced nodes (this substantially reduces the computational effort). Finally, for the simple baseline, we use **RND**, a model making uniformly random decisions. The number of parameters in each model (the size of X, §6.7.2) are 12.5M for LT and ALO and 1.6M for LTr. The sigmoid parameters for LT are set to $a = 20$ and $b = 0.5$.

Training. We train all models starting from a random parameter initialization for 30 iterations of gradient ascent (§6.7.2). Training takes at most 20min on a single modern CPU core.

Metrics. Each model, after it has been trained, outputs p_i^u predictions in the range $[0, 1]$.

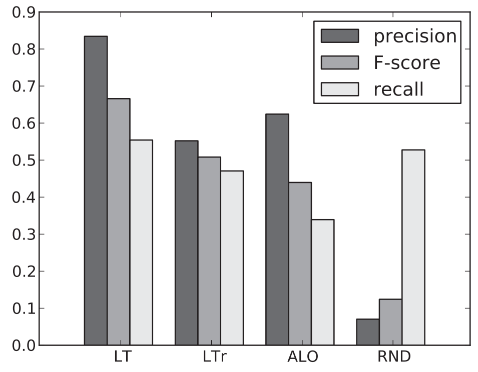


Figure 6.10. Performance of the models. For each model we measure the precision, recall and the F-score within the scope of the one-hop neighborhood of the nodes that have already mentioned the test URLs. The linear threshold model has the highest F-score and is able to predict more than half of the events (55% recall) at less than 15% false positive rate (85% precision).

We use the *discretization threshold* $\lambda = 0.5$ to classify the events into positive ($p_i^u > \lambda$) or negative ($p_i^u \leq \lambda$). For each u, we take the one-hop neighborhood of all the nodes that mentioned u. Within the scope of one-hop we compute the precision, recall and the F-score based on the discretized predictions and the true events from the test data set.

6.8.2. Results

Precision, recall, F-score. Figure 6.10 presents the precision, recall and F-score values for the one-hop neighborhood.

We observe the linear threshold model (LT) is able to correctly predict almost half of the URL mentions (55% recall) with at most 15% of false positives among the predictions (85% precision). It also gives the highest F-score out of all the models. Although negatives dominate the training and test data sets, LT learns the parameters affecting the positives well. The recall indicates that the model is unable to effectively learn the parameters for the 45% of positives in test set. These positives can most likely be explained by some other factors that the model does not account for. These factors are are most possibly external in nature as Twitter is a part of a much larger URL sharing ecosystem.

Controlling the precision-recall tradeoff: Some applications may require a higher precision from the model. The precision-recall tradeoff is controlled by varying the discretization threshold. The tradeoff for the LT model[4] is illustrated in Fig. 6.11. After the initial sharp increase in precision and decrease in recall, precision continues to increase without compromising on recall. Given that negatives dominate in our training data set, it underlines the ability of the model to learn the positives well and at higher thresholds it removes only the negatives,

[4] The curves for other models were similar

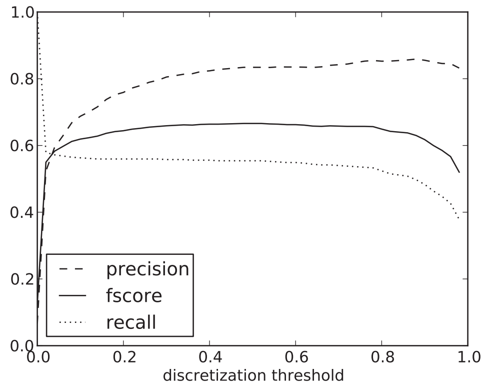


Figure 6.11. The precision-recall tradeoff. By adjusting the discretization threshold, the recall can be traded off for precision if the application requires it.

further improving precision. We are currently exploring alternative techniques for increasing the recall.

6.9. Discussion

The information propagation laws. In our 300 hour data set of 15M tweets we have observed several strong regularities. The user activity and the frequency of URL mentions are power-law distributed. The URL cascades are shallow with exponentially falling off height. They are composed of subcascades whose both number and size follow power-law distributions. We have also found that the diffusion delay follows the log-normal distribution extremely well.

Predicting the next hop and beyond. Our propagation model (§6.7) specifies the tweeting probabilities p_i^u as the function of the tweeting probabilities of the neighbors (p_j^us). However, during training that dependency is removed by substituting the neighbor p_j^us with the known values from the training data set. In this way, the set of p_i^us that need to be computed is determined by the training data set and the evaluation (§6.8) shows that if the training and prediction scope is narrowed down to the followers of the users that tweet URLs, our model predicts more than half of the events with less than 15% of false positives. Taking into account the empirical observation that most cascades are shallow (§6.6), the one hop neighborhood is likely to include most of the new URL mentions.

To be able to predict further than one hop away, the model would need to include p_i^us for more (u, i) pairs, however given the large number of URLs and the social network fan-out, this quickly becomes a computationally infeasible problem even for a two-hop neighborhood.

Another challenge is predicting not only the spread of the URLs but also the appearance of new root nodes initiating new subcascades. This requires a more detailed analysis of the conditions under which these events happen to be able to model them accurately.

Training. We have used the gradient ascent method for finding the set of model parameters under which the F-score is maximized. There is no guarantee that the global maximum has been reached. However, we have run training with different random seeds and arrived at models with identical (high) performance. Training could potentially be recast as another known combinatorial optimization problem, within which more precise optimality arguments could be made.

Continuous model updates. Twitter is a real-time medium with a continuous stream of URL tweets. Keeping the model up-to-date while the new data is arriving is a separate and challenging problem. The iterative training algorithm we are using could be running constantly, following the changing optimal point in the parameter space. As new users and URLs appear, their corresponding parameters could be trained separately and more aggressively for faster alignment with the rest of the model. The training algorithm is embarrassingly parallel, composed mostly of large sums and can readily be scaled on frameworks such as e.g., MapReduce.

6.10. Conclusions

We have tracked the spread of 15M URLs over a 300 hour period on the Twitter microblogging service. Several statistical regularities are discovered: the power-laws in user activity, the exponentially falling off cascade depth, the decomposition of cascades into subcascades and finally the log-normal distribution of the diffusion delay. We also propose an information propagation model that predicts over half of the future URL mentions, while having only a 15% false positive rate. The model can serve as an important building block for several applications: personalized URL recommendation, filtering of incoming tweets and spam detection, problem areas we hope will attract future research.

Our results show that the past information forwarding behavior is a reliable predictor of the future behavior. This is also true of the two routing protocols, FFP and Castor, that we have presented in the previous chapters. In both protocols, the past routing reliability of the node's neighbors is a good predictor of their future reliability. The nodes take advantage of this property for making correct routing decisions and ensuring the robustness of the information flows. Both protocols explicitly signal information delivery through either the feedback messages in FFP or the ACKs in Castor. On Twitter, the retweets play the role of the feedback and, as the results of this chapter indicate, they play an important role in reinforcing the paths along which the information flows in the social networks.

Publications: [158]

Chapter 7

Leveraging Social Networks for Increased Content Distribution Robustness

7.1. Introduction

In the previous chapter, we have examined the information flows in the on-line social network of Twitter. Despite the large scale of this network, its operation is mostly cooperative [141]. In systems based on the social networks, the identities of each of the nodes are tied to the real-world identities of a particular person or an organization, which creates a disincentive for the non-cooperative behavior. This is in contrast to the peer-to-peer content delivery systems such as BitTorrent [31], where the nodes join the system anonymously or with weak pseudonyms, which leads to a case where peers have little prior knowledge about one another and cannot assume that the others will always cooperate. BitTorrent peers can, for example, choose to download files but not upload any to conserve bandwidth. To prevent the freeriding behavior, additional mechanisms have to be built into the system. The design of such mechanisms is a non-trivial problem and has received considerable attention [159, 160, 161, 162]. We propose an alternative solution to the problem based on the following observation.

In most of the widely-deployed P2P content delivery systems, a single peer usually corresponds to a single user. This leads to an observation that it is not the peers in the peer-to-peer system that are selfish, but the human users of the peers. It is the users themselves who decide how their nodes should behave, either by making software or hardware changes. Hence, we could conjecture that a number of the social mechanisms described in §2.4.2.5 are also driving the evolution of peer behaviors in content distribution networks.

This conjecture is corroborated by the following anecdotal evidence. A peer-to-peer file sharing software called eMule[1] is open source, which allows anyone to make modifications to it

[1] `http://www.emule-project.net/`

and distribute them. This has given rise to a number of mutated versions of the base eMule client, the so-called "mods"[2]. There are mods that protect the user privacy by encrypting downloaded data, there are mods which implement various bandwidth saving heuristics, there are extremely non cooperative mods that cut off uploads to other peers to conserve bandwidth, there are mods that detect non-cooperative mods and disconnect from them, there are even mods that detect those policing mods and use stealth techniques to hide their defection etc. These mods are constantly created and propagated via numerous websites and evaluated by users on various on-line forums. The social network of peer-to-peer system users selecting behaviors for their peers is tightly interrelated with the peer-to-peer overlay network providing an arena for the execution of those behaviors selected by the users. *The social network external to the system acts as the mechanism for enforcing cooperation within it.* This is the key concept that we rely on in this chapter for freeriding prevention.

Given the popularity of some of the P2P applications, it is very likely that for most users a significant fraction of their social network friends also use the same application. Most P2P content delivery systems do not take advantage of these social links.

In a *social P2P system*, communication occurs almost exclusively between nodes whose owners have a trustworthy social relationship. These systems are also known as darknets [163, 164] or friend-to-friend (F2F) systems [165]. The basic assumption behind such systems is that just as people interacting in the real world are much less likely to cheat against friends than strangers, peers in an F2F system are likely to behave cooperatively towards other peers knowing that they belong to friends. There is thus little need for additional cooperation-enforcing mechanisms. This simplifies the system design.

In this chapter we evaluate a socially-aware BitTorrent, in which peers connect both to the peers obtained from the trackers and to the peers belonging to the user's friends. The three core problems that we address are: (1) whether the social network topology alone can function as a scalable and efficient content distribution medium, (2) whether we can take advantage of the social links for solving the freeriding problem without sacrificing performance and (3) what is the number of downloaders in the social network at which the benefit of using the social links becomes significant.

In the first system variant we consider communication occurs strictly only along the social links. The social network topology alone turns out to be an efficient content distribution medium (§7.4.2) with the same system scaling properties as the tracker-based BitTorrent (§7.4.6). Only a small poorly connected minority of the peers finish late, which is unavoidable due the social networks' power-law node degree distributions [166].

We also examine the BitTorrent's tit-for-tat (TFT) mechanism [31] in the presence of freeriders. As in the other studies [159, 160, 161, 162], our measurements confirm that TFT is not an effective solution to the freeriding problem. On the other hand, the social BitTorrent turns out to be remarkably robust to freeriding (§7.4.2). There is only a slight increase in completion

[2] `http://www.emule-mods.de/`

times even as 33% of the peers are freeriders, while in the classical tracker-based BitTorrent, the impact of the freeriders on performance is much more significant.

One caveat of relying only on the social links for content distribution is that not all the friends in the social network might be interested in downloading the same content. If only the social links are used for the downloads and the content is rare, not all the peers might find download partners among their friends and may not be able to complete the download. For rare content, peers must thus still rely on the trackers to discover the other peers interested in the same content. We evaluate a hybrid design in which peers use both their friends and the peers from the trackers for downloads. We show that in such a system, even if the content is rare the social links still improve the robustness to freeriding (§7.4.4). The social BitTorrent's downloads are faster than regular BitTorrent's up until the fraction of downloaders in the social network falls below 2%. The 2% threshold clearly delineates the setups in which the social networks can be leveraged to combat the freeriding in BitTorrent. Our measurements additionally show that apart from solving the sparse swarm problem, the hybrid solution has the most balanced upload bandwidth utilization (§7.4.5) among all the evaluated system variants.

7.2. Related work

There are several P2P systems that use the concept of social networks in one way or another. Li et al. [165] propose an F2F system for scalable and durable storage. The Freenet project [34] turns a social network into a privacy-preserving file-sharing application with focus on efficient search [167]. Tribler [168], another file-sharing application, allows the creation of social groups. Peers can recruit their friends from the group as download helpers to improve the performance. The helpers contribute their bandwidth twice, downloading from the source and then uploading to the friend that requested help. In contrast, in our system there is no recruitment but we do take advantage of the fact that two friends are downloading the same content. None of the above work looks at the social network topology as the sole means of content distribution and does not clearly measure the performance benefits gained from using the social links.

The popular BitTorrent client, Vuze, has recently added the friends feature[3]. Users can allocate more upload bandwidth to their friends and easily share torrent links with them. The eMule client [169] has friend lists as well. Users can allocate upload slots to friends so they do not need to wait in the queue. However, up to our knowledge the performance impact of the friend slots has not been investigated for large-scale BitTorrent swarms, we do that in this chapter.

The original solution to the problem of freeriding in BitTorrent is the tit-for-tat algorithm which has been shown to have limited effectiveness [159, 160, 161, 162]. Several other solutions have been proposed. Locher et al. [170] use coding to increase block diversity in the network and change the seeder strategy to only upload a globally fixed set of blocks per leecher. This

[3] `http://azureus.sourceforge.net/`

forces the freeriders to upload blocks to other peers in order to obtain new blocks in return. In the Piatek's et al. approach [171], peers share download accounting information (bytes sent, received etc.) of their neighbors with the other neighbors. This one-hop reputation system enables indirect reciprocity as opposed to TFT's pairwise reciprocity which results in increased performance. We propose to solve the problem of freeriding by taking advantage of the social links, for which cheating is unlikely. Each peer's identity is tied to a real person's identity and we, in a way, use the external "human reputation system" for ensuring cooperative peer behavior (§7.3.3).

Social networks can be used to improve the security and privacy of peer-to-peer systems by constraining the communication only to the social links and cryptographically protecting the exchanged data [164]. Friends can authenticate one another, for example with PGP or other methods [172]. One such system is OneSwarm [173], a friend-to-friend content delivery system based on the Azureus BitTorrent client that leverages the social network of Google Talk users to build an F2F system with the goal of inhibiting the monitoring of user behavior by external parties. The nodes communicate strictly only along the social links. Content search is done by flooding the social network with requests. The returning search responses build paths in the network along which the content is then forwarded hop-by-hop. This results in a much higher bandwidth consumption, proportional to the number of hops. In contrast, in our system all content is shared in a single hop as in the original BitTorrent and we take advantage of the social network to solve the freeriding problem and do not address the privacy or security.

7.3. System architecture

Making BitTorrent social requires only a few changes. In particular, the content distribution protocol does not have to be modified in any way, we only change the way in which peers discover one another. We also leave the standard mechanisms unchanged, such as the periodic chokes/unchokes, the end-game protocol and optimistic unchoking.

7.3.1. Friend connections

Since the social network topology is the integral part of the system we need to provide the users with a way to manage their social links (friend lists). To create a new social link the two users exchange cryptographic information that is later used for mutual authentication. Existing infrastructures such as PGP or F2F-specific protocols [172] can be used to facilitate that. For additional security the friend-to-friend communication can be encrypted.

When a peer starts up, it needs to be able to find the peers belonging to the other friends and connect to them. There exist several ways of achieving this, most of the instant messaging clients have solved this problem (e.g., the open decentralized Jabber/XMPP[4] infrastructure).

[4] `http://xmpp.org/about/jabber.shtml`

The social BitTorrent could also be integrated with the existing social network platforms using their APIs, e.g., Facebook[5]. In our evaluation we simply use a single instance of a friend server through which the peers discover one another.

The connections to friends are added to the local peer list and the standard BitTorrent protocol is executed as if these peers were discovered through a tracker.

7.3.2. Friend upload priority

In the social BitTorrent the friend connections are treated specially. Every time the upload slots are allocated, which happens periodically every 10s in our implementation, the friend connections take absolute priority over the other connections irrespective of their upload and download histories. For other, non-friend connections we are still using the **tit-for-tat (TFT)**, i.e. upload slots are allocated to the peer that is uploading the most.

7.3.3. Freeriding disincentive

The users are made aware of the friend connections and all the activity occurring on them. The status of the friend connections and the friend identities are displayed on the user interface. If some friend refuses to upload anything or artificially caps the upload bandwidth, it is made clearly visible on the user interface and the identity of the cheater is known. We assume throughout this chapter that this provides a strong disincentive for freeriding on friends, but does not prevent the users from freeriding on non-friend peers.

7.4. Evaluation

We have implemented the social BitTorrent in the ProtoPeer framework (Appendix A) based on the unofficial BitTorrent specification[6] complete with the trackers, the tit-for-tat mechanism, optimistic unchoking, partially downloaded piece prioritization and the end-game strategy.

We use the flow-based network model[7] in ProtoPeer which assumes a fixed upload bandwidth cap for each peer and divides the bandwidth equally among the outgoing flows. For simplicity, we assume the download bandwidth to be unbounded, since the upload bandwidth is typically the bottleneck in file-sharing systems. To introduce diversity, we assume that 10% of the peers have a high 2Mbit/s upload bandwidth, 30% are with 1Mbit/s and the rest with 512kbit/s, which is loosely based on the residential broadband measurements in [174]. In all the experiments we assume that initially a single source has the complete file. The source has a 2Mbit/s upload bandwidth.

[5] `http://developers.facebook.com/`
[6] `http://wiki.theory.org/BitTorrentSpecification`
[7] `http://protopeer.epfl.ch/wiki/FlowBasedNetworkModel`

The social network topologies for our evaluation were generated based on node degree and clustering coefficient distributions of an actual large-scale social graph of a popular instant messaging client [166]. The node degree is bounded by 64 and the graph is connected.

Throughout our evaluation we assume that the peers arrive in the system within the first 10 seconds and never depart. We have determined in simulations that peer arrivals and departures affect the social BitTorrent in the same way as the regular one and we omit these results.

Unless otherwise stated, the downloaded content is a 25MB file divided into 50 512kB pieces, each consisting of 32 16kB blocks. All peers use the rarest first piece selection strategy. The network consists of 256 nodes including the initial source. We vary the network size and the content size in separate scalability tests (§7.4.6).

The ***freeriders*** in the system are defined as peers who ignore all requests for content, but still request the content from the other peers. The freeriders are chosen in the system uniformly at random.

7.4.1. The lineup

We consider the following systems: **BT** - the classical BitTorrent system using a centralized tracker to discover 25 random peers to connect to, **SBT** - the strictly social variant of BitTorrent running on top of the generated social network, the data flows only along the social links, no trackers are used and **HBT** - the hybrid BitTorrent system, which uses both the social links and the trackers to discover additional uploaders as in the original BitTorrent.

7.4.2. Completion time

The most important performance metric of a file-sharing system is the time it takes to complete the file download on all the peers (Fig. 7.1).

Content distribution in the social network is not significantly slower than in the open tracker-based system. However, SBT suffers from the long-tail problem: the last $20\% - 30\%$ of the peers complete their downloads much later than the majority. Social graphs have a power-law degree distribution [166] with many peers having only a few links. It is those peers that cause the problem, they do not have enough peers to download from and the whole download process is delayed.

The set of results on Fig. 7.1 also illustrates the superior robustness to freeriding of the social BitTorrent. When freeriders are present, the open system (BT) is slower than the social system (SBT) for the majority of the peers, even when tit-for-tat is enabled.

7.4.3. Sparse swarms

Up to now we have assumed that all the peers in the social graph are interested in downloading the same file. This is not likely in reality. Peers that are not interested in the file do not download or upload any data. Let ***swarm density*** be the fraction of peers in the social

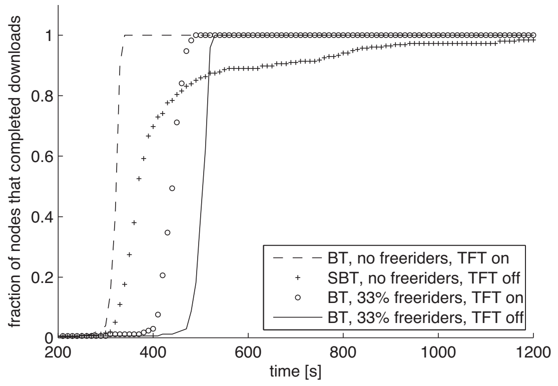


Figure 7.1. Completion times. A 25MB file is distributed from a single initial seeder using the BitTorrent system (BT) and social BitTorrent (SBT). We measure the download completion times at non-freerider peers. The first 70% of the peers in SBT complete their downloads 1 minute later than in BT. The next 30% of the peers in SBT are mainly the nodes that are poorly connected and require more time for the download. When every third peer is a freerider in BT, the downloads slow down by 200s. Enabling tit-for-tat (TFT) improves the performance, but BT is still slower than SBT.

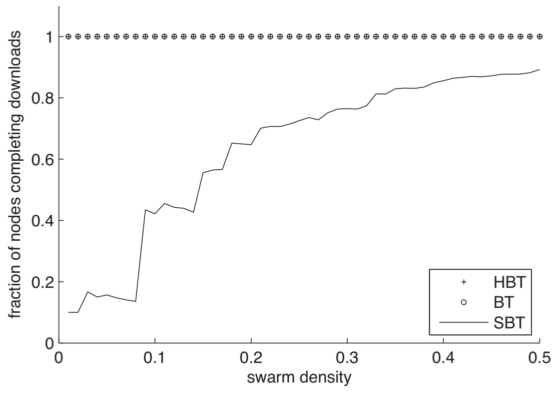


Figure 7.2. Sparse swarms. At low swarm density, i.e., when only a few of the social network nodes are interested in the download, not all the downloaders can access a seeder via a chain of uploaders and complete their downloads.

network that are interested in the file. If the density is low then some interested peers may not be able to reach any seeder via a chain of interested peers and will not complete their downloads. We have confirmed that this effect is in fact severe (Fig. 7.2).

To solve the sparse swarm problem the peers need some method of finding other peers interested in downloading the same content. This can either be done in a decentralized way by e.g., flooding the social network with search requests (as in OneSwarm [173]) or by using the trackers as in the original BitTorrent.

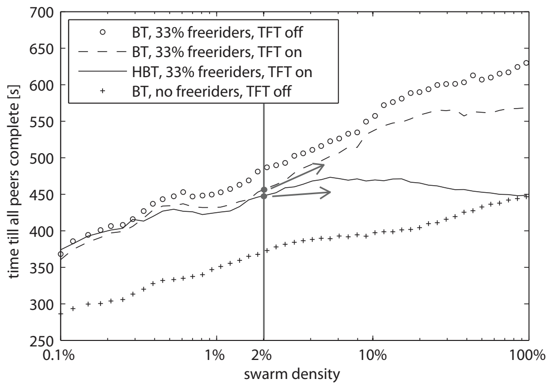


Figure 7.3. Varying the swarm density. In HBT, peers use both the peers discovered via the tracker and their direct friends as the uploaders. The BitTorrent systems without TFT and without freeriders are included as the baselines. In all systems the social network size is 10000 and the file is 25MB. The performance improvement gained by using the social links (HBT) is the bigger the denser the swarm is. However, when the density falls below 2% (i.e., less than 200 downloaders among the 10000 peers), the performance advantage offered by the social links disappears.

7.4.4. Adding the trackers - the hybrid system

To complete their downloads in a sparse swarm, peers can discover additional uploaders through a tracker. At the same time, the peers can still use their friends as reliable uploaders when possible. We look at how the performance of such a hybrid system (HBT) changes with the swarm density (Fig. 7.3).

In high density swarms, peers have no trouble finding friends interested in the same file. As the swarm density decreases, peers can no longer rely on downloads from friends and have to download from strangers, which can potentially be freeriders. When the swarm density falls down to approximately 2% the performance advantage of the social BitTorrent disappears, the majority of downloads are from strangers. This clearly shows in which setups the social networks can be used to improve BitTorrent's robustness to freeriding.

7.4.5. Bandwidth utilization

Next we look at how balanced the bandwidth utilization is in the system (Fig. 7.4). In the case of BT, the 33% of the freeriders are clearly visible, they do not contribute any bandwidth. In the case of SBT, a few socially well connected nodes contribute a disproportionate amount of bandwidth to serving the poorly connected nodes. The hybrid system (HBT) is the most balanced, poorly connected nodes obtain additional download sources from the trackers, while the majority of the freeriding is prevented by leveraging the social links.

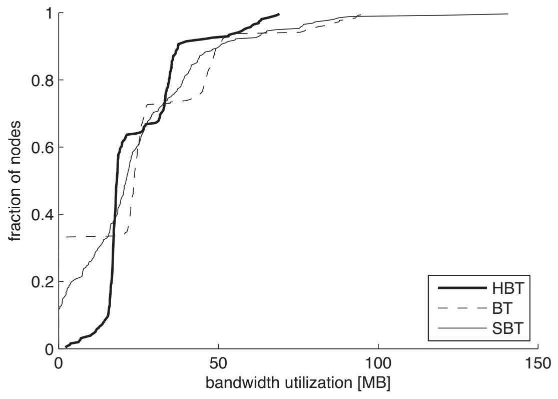
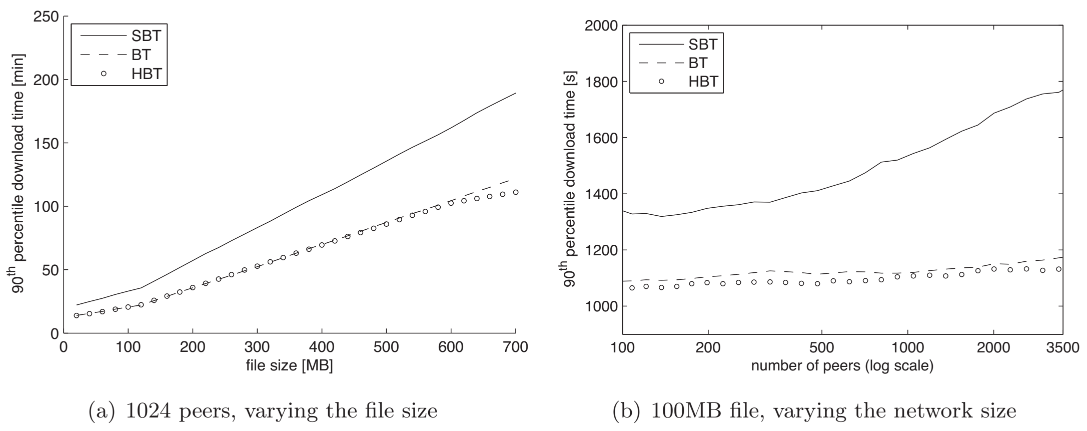


Figure 7.4. CDF of the bandwidth utilization. A 25MB file is distributed from a single initial seeder in the regular (BT), the social (SBT) and the hybrid (HBT) BitTorrent system. Every third node is a freerider, tit-for-tat is enabled. We measure the cumulative distribution function of the total upload bandwidth used per node during the file distribution. The area to the left of the curve remains constant across the systems, the same amount of data is uploaded in total. The closer the curve is to a vertical line the more balanced the bandwidth utilization is. Note: 10% of the peers have a 2Mbit/s upload bandwidth, 30% 1Mbit/s and the rest 512kbit/s, which is visible on the plots as "steps".



(a) 1024 peers, varying the file size (b) 100MB file, varying the network size

Figure 7.5. System scaling. Every peer in the social network is interested in the file. We vary the network and file size and measure the 90^{th} percentile of download times. All peers are active downloaders. The scaling is linear in the file size and logarithmic in the network size. The system scaling properties of the strictly social BitTorrent are not significantly different than those of the regular tracker-based system.

7.4.6. Scalability

In all our previous experiments the file size and the network size were fixed. We now examine how the performance changes as these parameters change (Fig. 7.5).

The results show that the download times scale linearly with the file size and logarithmically with the number of peers in the network. This is in line with the scaling formulas derived in [175] for the optimal BitTorrent content distribution. The social network alone (SBT) turns out to be just as scalable as the classical solution based on the centralized tracker (BT), albeit the completion times are slower due to the power-law degree distributions of the social graphs (§7.4.2).

7.5. Discussion

Social network as a content distribution medium. We have shown that the topology of the social network alone without relying on the trackers can function as an efficient and scalable content distribution medium. For the majority of the nodes in the strictly social BitTorrent using only direct friends as uploaders the completion times are as short as in a BitTorrent system using trackers to interconnect the peers. However, the peers with fewer friends experience longer completion times and to improve their performance need to find additional uploaders. The best uploaders are simply new friends. This creates a strong incentive for the friends to connect to other friends and could potentially drive the growth of BitTorrent's social network.

Robustness to freeriding. The BitTorrent's tit-for-tat mechanism is known to be an ineffective solution to the freeriding problem [159, 160, 161, 162]. We have confirmed this in our measurements (§7.4.2). When freeriders are present, enabling tit-for-tat increases the performance, but only slightly. This ineffectiveness, as others have observed [176], is mainly attributable to the altruism of the seeders, which unchoke downloaders without regard to their past upload performance.

We have confirmed experimentally that the reliable social links are crucial to increasing the robustness to freeriding. Even when 33% of the additional uploaders are freeriders the performance of the social BitTorrent drops only insignificantly. The social network forms a backbone for reliable cooperative content distribution and the freeriders' impact is lowered.

Best of two worlds: the hybrid system. The BitTorrent's tit-for-tat mechanism does not prevent freeriding, but merely provides a disincentive for it. The freeriders do not contribute any of their upload bandwidth and, as we have measured in §7.4.5, the downloaders must download from other peers, which creates a bandwidth utilization imbalance in the system.

In the BitTorrent variant that uses only the social links for downloads (SBT), a small fraction of the peers are poorly connected and use the highly connected peers as download hubs. This is a fundamental problem caused by the power-law node degree distributions of the social graphs [166]. Again, there is a bandwidth utilization imbalance in the system.

As we have shown, the above two approaches can be combined into one (HBT), in which peers connect to both their friends as well as strangers discovered via the centralized trackers. This results in a system with a balanced bandwidth utilization, high robustness to freeriding and short download completion times.

Real-world swarm densities We have shown in §7.4.4 that once the swarm density falls below 2% the social BitTorrent can no longer rely on the social network for combating the freeriding. This brings up important questions. Is 2% a high or a low value? What are the actual swarm densities in the social networks? We are not aware of any work addressing this question. We can, however, offer some hypotheses. First, in our experiments we have assumed that the peers that are part of the swarm are uniformly randomly distributed in the social network. This is rather unrealistic. Friends share common interests and are likely to be interested in the same files, peers belonging to the same swarm would tend to cluster together in the social network. Second, our social BitTorrent is not the only content distribution network, it interacts with the others out-of-system. It is very likely that not one, as we assumed, but several initial sources of the file would appear in the network. This could significantly affect the ability of the peers to download the file in the tracker-less variants of the social BitTorrent. In the absence of any real-world data we were unable to experimentally determine how the above effects would influence the performance of our system.

Actual user behavior. We and the designers of the other friend-to-friend systems [165, 164] have assumed that peers are less likely to cheat against their friends than strangers. Up to our knowledge there are no measurements from widely deployed friend-to-friend systems that can confirm that. There is also an interesting side question: Are the friends of friends as reliable as the direct friends? We have assumed in the evaluation that this is not the case. However, in reality some freeriders might still be cooperative towards the friends of friends knowing that they are not complete strangers. This might further increase the robustness of our system to freeriding.

Beyond freeriding robustness. The advantages of socially-aware content distribution systems go beyond improved robustness to freeriding. As we have shown, the social network alone without any centralized components can be turned into a scalable and efficient content distribution medium. Both the decentralization and the fact that communication can be constrained only to the social links significantly increase the security and privacy beyond what is available in the current popular systems.

7.6. Conclusions

We have presented a version of BitTorrent that takes advantage of the social links for cooperative content distribution. Peers are unlikely to cheat against their real-world friends, which substantially increases the system's robustness to freeriding. The swarms must be sufficiently dense for this effect to become noticeable, although the critical swarm density is surprisingly low - only 2%. The hybrid version of our system has the best overall performance. It opportunistically uses the friend's peers to ensure cooperative reciprocal downloads and falls back on the less reliable peers obtained from the trackers if the swarm density is too low. The trust latent in social networks could potentially be leveraged to solve cooperation problems in other distributed systems and hybrid socio-technological designs are a promising direction of research.

Publications: [177, 178]

Chapter 8

Outlook & Conclusions

For networks to support robust information flows, a number of design challenges must be addressed. First, the network's topology must be small-world such that each node can be reached from the other nodes efficiently. Second, as nodes join and leave and as the links and nodes fail, the network must adapt its topology to ensure the reliability of the information flows. Finally, the resource constraints at each node might cause the autonomously operating nodes to exhibit selfish behavior, which needs to be counteracted with cooperation-enforcing mechanisms. We have addressed these problems across four different types of networks: peer-to-peer overlays, mobile ad-hoc networks, social networks and file-sharing networks.

8.1. Network formation in arbitrary metric spaces

We first tackled the problem of constructing a general network formation and routing algorithm that given a set of nodes embedded in an arbitrary metric space would wire them in a topology that supports efficient information flows (§3). The algorithm that we proposed was applied in peer-to-peer systems for overlay routing. We have demonstrated that our solution has desirable scalability properties and tolerance to node churn in a wide variety of metric spaces. The ability to customize the metric space gives the applications running on top of P2P overlays great flexibility on how they can place their data and computation tasks in the distributed system.

Moreover, our work by proposing a concrete network formation protocol for nodes embedded in arbitrary metric spaces provides additional support for the hypothesis that greedy routing and metric space embedding could be the universal components of routing mechanisms in the real-world complex networks [1].

In our protocol, a node sends a connection request when the lookup message that it is routing has not been making progress towards the destination fast enough. This is governed by a strict rule. However, this rule could be relaxed in many ways leading to several new applications of the protocol. Firstly, each node could autonomously decide whether it should respond to the connection request. For example, if a node is bandwidth-constrained, it may accept fewer connections and in consequence forward less lookup messages. The connection requests would be passed on to the node's neighbors. Secondly, there may be constraints on which nodes can connect to which nodes in the network. For example, on the Internet, many nodes are behind firewalls and may not be able to accept connections. The network formation should be designed to tolerate such cases. Finally, the protocol could take the network delay into account and selectively form connections only between those nodes that are close in the each other in the Internet topology, e.g., in the same autonomous system.

8.2. Robust information flows

Constructing a topology that supports efficient routing is not sufficient. Routing needs to be made robust to message loss and delays that occur at shorter time-scales than node joins an departures. Solutions to this problem that send multiple copies of the same message to ensure timely message delivery consume excessive amounts of bandwidth. Moreover, as the network grows in scale, it needs to handle frequent topology changes. To keep the routing state up-to-date and path reliability, state-of-the-art protocols exchange routing information. Besides substantially contributing to the bandwidth overhead in large networks, passing the routing information between the nodes introduces a wide range of security problems, mainly centered around ensuring the integrity and authenticity of the exchanged routing information.

To address these problems, we proposed two variants of the same approach to robust routing: the Forward Feedback Protocol (§4) in peer-to-peer overlays, and the more secure Castor protocol for mobile ad-hoc networks (§5). Both protocols follow the similar pattern. For bandwidth efficiency, the messages are routed along a single path only. Through simple signaling, the system learns which peers to avoid and unlike other approaches, continuously learns from its routing mistakes and rapidly adapts to the changing delay and loss conditions in the network. Our measurements have confirmed that both protocols consume less bandwidth than the state-of-the-art, while at the same time having similar or better delivery rates. In the case of Castor, because it relies on simple signaling, both the message exchange and the in-network state are easier to secure.

There are several important characteristics that both FFP and Castor share. Firstly, in both protocols, nodes have a significant autonomy. Each node independently observes the feedback signals it receives after forwarding the messages. The nodes never exchange any part of their routing state with the other nodes. This is key to robustness of the protocols. When a node starts failing or is controlled by an adversary, its incorrect routing state never escalates to the whole network. The anomaly is contained and routed around.

Secondly, both protocols are topology-oblivious, that is, each node only observes its own network neighborhood, either the currently connected peers in the case of P2P overlays or the nodes currently in the radio range in the case of MANETs. No node is aware of the complete network topology. By observing the feedback signals, all nodes learn over time which next hops are reliable for forwarding messages to which part of the network. Topology-obliviousness means that the processes of routing and route discovery are completely decoupled from the processes that form the topology. This suggests that our signaling-based approach to routing could be used in many other networks. One possibility is to combine FFP with the network formation algorithm proposed in Chapter 3 for improved robustness. Another possibility is to use our feedback-based technique for discovering reliable routes in other complex networks: matching up the publishers and subscribers in pub/sub systems, discovering people who might answer a question in social-network-based Q&A systems or improving the relevance of the information flowing in the on-line social networks (a problem that we addressed in Chapter 6).

Finally, the feedback in the two protocols signals whether the routing succeeded or not. However, the signal could be made to mean anything else depending on the application. In particular, any information that could help make the nodes more informed routing decisions might lead to higher performance protocols. For example, signals could indicate network congestion on the routing path or the total delay to the destination. This would allow the nodes to optimize for other metrics besides reliability.

8.3. The security-scalability tradeoff

The vulnerability of the Forward Feedback Protocol to malicious message injection is addressed in Castor by cryptographically ensuring the integrity and authenticity of the messages. Moreover, key to Castor's security is the separation of the in-network routing state for each flow. Flows injected into the network by the attacker cannot influence the routing state of the other flows. Keeping per-flow state at the nodes may work at the scale of the hundreds of nodes in MANETs, but would not scale to the sizes of Internet-wide peer-to-peer overlays, where each node is involved in forwarding many flows (or distinct source-destination pairs).

To address the scalability problem, FFP aggregates the routing state at each node into zones (4.3.3), which results in an exponential reduction in the per-node state size compared to the case where the state is kept per-flow or per-destination. However, such state aggregation, if implemented in Castor, would severely decrease its security as the attacker would be able to disrupt the benign flows by injecting fake flows and, in effect, maliciously modifying the shared aggregate routing state. The differences in Castor's and FFP's routing state management reflect the fundamental security-scalability tradeoff faced by the information flow design in large distributed systems.

8.4. Information flows in on-line social networks

Chapters 4 and 5 have examined the routing protocols in peer-to-peer and mobile ad-hoc networks. The routing decisions are based on the history of received feedback signals corresponding to the forwarded messages. In Chapter 6 we have examined the signaling by "retweeting" messages in the online social network of Twitter. In FFP and Castor, the history of signals for a given node neighbor is a good predictor of the future success of routing. In Twitter's case, the history of retweets is a good predictor of the future information flow in the social network. The model that we have proposed successfully accounts for more than half of the URL mentions on Twitter with the false positive rate as low as 15%.

The ability to accurately predict which user will mention what information has many applications. Personalized content filters can be built by displaying only the information that users are most likely to forward. Interesting content can be surfaced early on, when the model already predicts the content's high potential to spread after only a few mentions. Finally, viral marketing campaigns can use the information flow models to predict the campaign's success based on which users will be the initial spreaders.

Finally, the model that we proposed is universally applicable to most of the existing on-line social networks, not only Twitter. Most of the networks have some form of simple signaling such as the "likes", "diggs" or "voting up". Although the dynamics of spread in other networks might be different, given its large parameter space, our model might potentially be trained to make accurate information flow predictions in those networks as well.

8.5. Socio-technological solution to freeriding

Large-scale complex networks are usually composed of autonomous nodes, each operating under its own resource constraints. To gain benefit from using the network, the nodes must typically expend their own resources to the benefit of the other nodes. However, the nodes might freeride, i.e., use the network and not contribute any resources to its operation. This problem is particularly challenging in peer-to-peer file-sharing networks. Both the scale of these systems and the lack of verifiable node identities make many of the purely technological solutions to this problem infeasible (§2.4).

In Chapter 7, we propose a solution based on Friend-to-Friend computing, in which the connections between two peers are only opened if their owners know one another in the real-world. This, in effect, maps the social network onto a computer network. We have demonstrated that a social network running the BitTorrent protocol is an efficient content distribution medium and the freeriding problem is solved by relying on the socially-enforced cooperation rather than only the technological means. In our evaluation, the best performing system was the hybrid one, in which the peers were connected to both untrusted peers discovered via BitTorrent trackers as well as the trusted socially discovered peers.

We have only examined one application, file-sharing, but leveraging the trust latent in the social networks could potentially be applied to solving other cooperation problems in peer-to-peer systems. For example, in a peer-to-peer overlay, the users could keep tabs on how many messages their peers have routed for one another. The software would attempt to reciprocate towards the other peers and alert the user if the other peers are not cooperative. Similarly, in the distributed hash tables, the users could keep track of how many keys each peer is responsible for and based on that socially enforce the compliance with all the data operations on the hash table.

8.6. Final words

In this thesis, inspired by the complex networks research, we took a very general point of view at reasoning about the network structures and the processes that occur on top of them. This allowed us to develop several protocols and algorithms that are not only applicable to networks that they were designed for, but also for others. Firstly, our overlay routing protocol (§3) is not only capable of forming efficiently routing peer-to-peer overlays, but could be applied in any other network where nodes use a metric space as their address space. It is particularly interesting whether the social networks or other real-world complex networks would be routable in this way once their underlying hidden metric space is uncovered. Secondly, we have designed a robust routing protocol that relies on simple signaling along the routing paths and applied it in two different networks (§4, §5). Again, given the general formulation of the protocol and its obliviousness of the topology, it could be applied to many other networks for route discovery and ensuring routing fault-tolerance. Thirdly, we looked at the spread of information in on-line social networks and how that spread is affected by the social signaling (§6). The model that we proposed can correctly predict the URL mentions on Twitter, but could potentially be applied to other on-line social networks or even predict the word-of-mouth information spread in the social networks unaugmented by technology. Finally, we have examined a system (§7) that combined two networks: peer-to-peer file-sharing network and the social networks. The two networks complement one another. One provides the massive communication bandwidth and the other the identities that make reciprocity possible and help build cooperation in the system. The success of this approach suggests that the social identities and trust are valuable assets that could be leveraged to solve cooperation problems in other distributed systems.

In closing, we can only hope that by taking such a general point of view on networks, this thesis will inspire more cross-disciplinary research into networks and processes ocurring on them as well as novel hybrid designs at the intersection of technological, biological and socioeconomic systems.

Bibliography

[1] Boguñá, M., Krioukov, D., Claffy, K.: Navigability of complex networks. Nature Physics **5**(1) (2008) 74–80

[2] Bondy, J., Murty, U.: Graph theory. Springer (2007)

[3] Albert, R., Barabási, A.: Statistical mechanics of complex networks. Reviews of modern physics **74**(1) (2002) 47–97

[4] Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., Hwang, D.: Complex networks: Structure and dynamics. Physics reports **424**(4-5) (2006) 175–308

[5] Newman, M.: The structure and function of complex networks. Arxiv preprint cond-mat/0303516 (2003)

[6] Cohen, R., Havlin, S.: Complex Networks: Structure, Robustness and Function. Cambridge Univ Pr (2010)

[7] Milgram, S.: The small world problem. Psychology today **2**(1) (1967) 60–67

[8] Newman, M.: The structure of scientific collaboration networks. Proceedings of the National Academy of Sciences of the United States of America **98**(2) (2001) 404

[9] Aiello, W., Chung, F., Lu, L.: A random graph model for massive graphs. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing, ACM (2000) 171–180

[10] Chen, Q., Chang, H., Govindan, R., Jamin, S.: The origin of power laws in Internet topologies revisited. In: INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Volume 2., IEEE (2002) 608–617

[11] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the web. Computer networks **33**(1-6) (2000) 309–320

[12] White, J., Southgate, E., Thomson, J., Brenner, S.: The structure of the nervous system of the nematode Caenorhabditis elegans. Philosophical Transactions of the Royal Society of London. B, Biological Sciences **314**(1165) (1986) 1

[13] Watts, D., Strogatz, S.: Collective dynamics of Ôsmall-worldÕnetworks. Nature **393**(6684) (1998) 440–442

[14] Rapoport, A.: Contribution to the theory of random and biased nets. Bulletin of mathematical biology **19**(4) (1957) 257–277

[15] Erdős, P., Rényi, A.: On the evolution of random graphs. Citeseer (1960)

[16] Price, D.: A general theory of bibliometric and other cumulative advantage processes. Journal of the American Society for Information Science **27**(5) (1976) 292–306

[17] Barabási, A., Albert, R.: Emergence of scaling in random networks. Science **286**(5439) (1999) 509

[18] Keeling, M., Eames, K.: Networks and epidemic models. Journal of the Royal Society Interface **2**(4) (2005) 295

[19] Hethcote, H.: The mathematics of infectious diseases. SIAM review **42**(4) (2000) 599–653

[20] Callaway, D., Newman, M., Strogatz, S., Watts, D.: Network robustness and fragility: Percolation on random graphs. Physical Review Letters **85**(25) (2000) 5468–5471

[21] Newman, M.: Spread of epidemic disease on networks. Physical Review E **66**(1) (2002) 16128

[22] Cohen, R., Havlin, S., Ben-Avraham, D.: Efficient immunization strategies for computer networks and populations. Physical Review Letters **91**(24) (2003) 247901

[23] Leskovec, J., Adamic, L., Huberman, B.: The dynamics of viral marketing. ACM Transactions on the Web (TWEB) **1**(1) (2007) 5

[24] Serenyi, D., Witten, B.: RapidUpdate: Peer-assisted distribution of security content. In: Proceedings of the 7th international conference on Peer-to-peer systems, USENIX Association (2008) 20

[25] Terry, D., Theimer, M., Petersen, K., Demers, A., Spreitzer, M., Hauser, C.: Managing update conflicts in Bayou, a weakly connected replicated storage system. ACM SIGOPS Operating Systems Review **29**(5) (1995) 172–182

[26] Jelasity, M., Montresor, A.: Epidemic-style proactive aggregation in large overlay networks. In: Distributed Computing Systems, 2004. Proceedings. 24th International Conference on, IEEE (2005) 102–109

[27] Watts, D., Dodds, P., Newman, M.: Identity and search in social networks. Science **296**(5571) (2002) 1302

[28] Kleinberg, J.: The small-world phenomenon: an algorithm perspective. In: Proceedings of the thirty-second annual ACM symposium on Theory of computing, ACM (2000) 163–170

[29] Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Computing Surveys **36**(4) (December 2004) 335–371

[30] Risson, J., Moors, T.: Survey of research towards robust peer-to-peer networks: Search methods. Computer Networks **50**(17) (2006) 3485–3521

[31] Cohen, B.: Incentives build robustness in BitTorrent. In: Workshop on Economics of Peer-to-Peer systems. Volume 6., Citeseer (2003) 68–72

[32] Ku, R., Shih, R.: The creative destruction of copyright: Napster and the new economics of digital technology. U. Chi. L. Rev. **69** (2002) 263

[33] Ripeanu, M.: Peer-to-peer architecture case study: Gnutella network (2001)

[34] Clarke, I., Sandberg, O., Wiley, B., Hong, T.W.: Freenet: A distributed anonymous information storage and retrieval system. Lecture Notes in Computer Science **2009** (2001) 46–53

[35] Aberer, K., Cudré-Mauroux, P., Datta, A., Despotovic, Z., Hauswirth, M., Punceva, M., Schmidt, R.: P-grid: a self-organizing structured P2P system. SIGMOD Record **32**(3) (2003) 29–33

[36] Stoica, I., Morris, R., Karger, D.R., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM'04. (2001) 149–160

[37] Maymounkov, P., Mazières, D.: Kademlia: A peer-to-peer information system based on the XOR metric. In: IPTPS. Volume 2429 of LNCS., Springer (2002) 53–65

[38] Lv, Q., Ratnasamy, S., Shenker, S.: Can heterogeneity make gnutella scalable? Peer-to-Peer Systems (2002) 94–103

[39] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications, ACM Press (2001) 161–172

[40] Fraigniaud, P., Gauron, P.: D2B: A de Bruijn based content-addressable network. Theoretical Computer Science **355**(1) (2006) 65–79

[41] Girdzijauskas, S., Galuba, W., Darlagiannis, V., Datta, A., Aberer, K.: Fuzzynet: Ringless routing in a ring-like structured overlay. Peer-to-Peer Networking and Applications (2010) 1–15 10.1007/s12083-010-0081-3.

[42] Gkantsidis, C., Mihail, M., Saberi, A.: Random walks in peer-to-peer networks: algorithms and evaluation. Performance Evaluation **63**(3) (2006) 241–263

[43] Wallach, D.: A survey of peer-to-peer security issues. Software Security - Theories and Systems (2003) 253–258

[44] Srivatsa, M., Liu, L.: Vulnerabilities and security threats in structured overlay networks: a quantitative analysis. In: Computer Security Applications Conference, 2004. 20th Annual, IEEE (2004) 252–261

[45] Dumitriu, D., Knightly, E., Kuzmanovic, A., Stoica, I., Zwaenepoel, W.: Denial-of-service resilience in peer-to-peer file sharing systems. In: SIGMETRICS '05, New York, NY, USA, ACM (2005) 38–49

[46] Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.: Secure routing for structured peer-to-peer overlay networks. ACM Operating Systems Review **36**(si) (2002) 299

[47] Atul Singh, Tsuen-Wan Ngan, P.D., Wallach, D.S.: Eclipse attacks on overlay networks: Threats and defenses. In: INFOCOM. (2006)

[48] Douceur: The sybil attack. In: International Workshop on Peer-to-Peer Systems (IPTPS), LNCS. Volume 1. (2002)

[49] Levine, B., Shields, C., Margolin, N.: A survey of solutions to the sybil attack. University of Massachusetts Amherst, Amherst, MA (2006)

[50] Christin, N., Weigend, A., Chuang, J.: Content availability, pollution and poisoning in file sharing peer-to-peer networks. In: Proceedings of the 6th ACM conference on Electronic commerce, ACM (2005) 68–77

[51] Tran, H., Hitchens, M., Varadharajan, V., Watters, P.: A Trust based Access Control Framework for P2P File-Sharing Systems. In: Proceedings of the Proceedings of the 38th Annual Hawaii International Conference on System Sciences-Volume 09, IEEE Computer Society (2005) 302–3

[52] Fiat, A., Saia, J.: Censorship resistant peer-to-peer content addressable networks. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics (2002) 94–103

[53] Giordano, S.: Mobile ad hoc networks. Handbook of wireless networks and mobile computing **1** (2002) 325–346

[54] Royer, E., Toh, C.: A review of current routing protocols for ad hoc mobile wireless networks. IEEE personal communications **6**(2) (1999) 46–55

[55] Buttyan, L., Hubaux, J.: Security and cooperation in wireless networks: thwarting malicious and selfish behavior in the age of ubiquitous computing. Cambridge Univ Press (2007)

[56] Mauve, M., Widmer, A., Hartenstein, H.: A survey on position-based routing in mobile ad hoc networks. Network, IEEE **15**(6) (2002) 30–39

[57] Fudenberg, D., Tirole, J.: Game Theory. MIT Press, Cambridge, MA (1991)

[58] Gambetta, D.: Can We Trust Trust? Trust Making and Breaking Cooperative Relations (1988) 213–237

[59] Aberer, K., Despotovic, Z., Galuba, W., Kellerer, W.: The Complex Facets of Reputation and Trust. In: 9th Fuzzy Days, International Conference on Computational Intelligence Fuzzy Logic Neural Networks Evolutionary Algorithms. Advances in Soft Computing, Springer (2006) 283–294

[60] Marsh, S.: Formalising trust as a computational concept. PhD thesis (1994)

[61] Despotovic, Z.: Building Trust-Aware P2P Systems: From Trust and Reputation Management to Decentralized e-Commerce Applications. Diploma thesis, EPFL, Swiss Federal Institute of Technology, Lausanne (2005)

[62] Despotovic, Z., Aberer, K.: A probabilistic approach to predict peers' performance in p2p networks. In: Eighth International Workshop on Cooperative Information Agents, CIA 2004, Erfurt, Germany (2004)

[63] Kreps, D.M., Wilson, R.: Reputation and imperfect information. Journal of Economic Theory **27** (1982) 253–279

[64] Kandori, M., Matsushima, H.: Private observation, communication and collusion. Econometrica **66**(3) (May 1998) 627–652

[65] Mailath, G.J., Morris, S.: Repeated games with almost-public monitoring. CARESS Working Papres almost-pub, University of Pennsylvania Center for Analytic Research and Economics in the Social Sciences (August 1999)

[66] Axelrod, R.M.: The Evolution of Cooperation. Basic Books (1984)

[67] Kandori, M.: Social norms and community enforcement. Review of Economic Studies **59**(1) (1992) 63–80

[68] H. Ohtsuki, Y.I.: How should we define goodness? reputation dynamics in indirect reciprocity. Journal of Theoretical Biology **231** (2004) 107–120

[69] Fabrikant, Papadimitriou, Talwar: The complexity of pure nash equilibria [extended abstract]. In: STOC: ACM Symposium on Theory of Computing (STOC). (2004)

[70] Conlisk, J.: Why bounded rationality? Journal of Economic Literature **34**(2) (June 1996) 669–700

[71] Trivers, R.L.: The evolution of reciprocal altruism. The Quarterly Review of Biology **46**(1) (Mar 1971) 35–57

[72] Nowak, M.A., Sigmund, K.: Evolution of indirect reciprocity. Nature **437** (2005) 1291–1298

[73] Robert Boyd, Herbert Gintis, S.B., Richerson, P.J.: The evolution of altruistic punishment. Proceedings of the National Academy of Sciences **100**(6) (Mar 2003) 3531–3535

[74] Heinrich, J., Boyd, R.: Why people punish defectors. J Theor Biol. **208**(1) (Jan 2001) 79–89

[75] Smith, E.A., Bowles, S., Gintis, H.: Costly signaling and cooperation. Working Papers 00-12-071, Santa Fe Institute (December 2000)

[76] Dierks, T., Rescorla, E.: Transport layer security over stream control transmission protocol. Technical Report 5246, The Transport Layer Security (TLS) Protocol (2002)

[77] Garfinkel, S.: PGP: pretty good privacy. O'reilly (1995)

[78] Datta, A., Hauswirth, M., Aberer, K.: Beyond Òweb of trustÓ: Enabling P2P E-commerce. In: IEEE Conference on Electronic Commerce (CECÕ03), Citeseer (2003)

[79] Zhong, S., Chen, J., Yang, Y.: Sprite: A simple, cheat-proof, credit-based system for mobile ad-hoc networks. In: INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies. Volume 3., IEEE (2003) 1987–1997

[80] Rivest, R.: Peppercoin micropayments. In: Financial Cryptography, Springer (2004) 2–8

[81] Rivest, R., Shamir, A.: PayWord and MicroMint: Two simple micropayment schemes. In: Security Protocols, Springer (1997) 69–87

[82] Vishnumurthy, V., Chandrakumar, S., Sirer, E.: Karma: A secure economic framework for peer-to-peer resource sharing. In: Workshop on Economics of Peer-to-Peer Systems. (2003)

[83] Garcia, F., Hoepman, J.: Off-line karma: A decentralized currency for peer-to-peer and grid applications. In: Applied Cryptography and Network Security, Springer (2005) 364–377

[84] Yang, B., Garcia-Molina, H.: Ppay: Micropayments for peer-to-peer systems. In: Proceedings of the 10th ACM conference on Computer and communications security, ACM (2003) 300–310

[85] Cox, L., Noble, B.: Samsara: Honor among thieves in peer-to-peer storage. In: Proceedings of the nineteenth ACM symposium on Operating systems principles, ACM (2003) 120–132

[86] Buttyán, L., Hubaux, J.: Stimulating cooperation in self-organizing mobile ad hoc networks. Mobile Networks and Applications **8**(5) (2003) 579–592

[87] Gupta, R., Somani, A.: Reputation management framework and its use as currency in large-scale peer-to-peer networks. In: Peer-to-Peer Computing, 2004. Proceedings. Proceedings. Fourth International Conference on, IEEE (2004) 124–132

[88] Zhao, B.Y., Kubiatowicz, J., Joseph, A.D.: Tapestry: a fault-tolerant wide-area application infrastructure. Computer Communication Review **32**(1) (2002) 81

[89] Rowstron, A., Druschel, P.: Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems. In: Middleware'01. (November 2001)

[90] Bharambe, A.R., Agrawal, M., Seshan, S.: Mercury: supporting scalable multi-attribute range queries. In: SIGCOMM '04: Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2004) 353–366

[91] Manku, G.S., Bawa, M., Raghavan, P.: Symphony: Distributed hashing in a small world. In: 4th USENIX Symposium on Internet Technologies and Systems, USITS. (2003)

[92] Bustamante, F., Qiao, Y.: Friendships that last: Peer lifespan and its role in p2p protocols (2003)

[93] Li, J., Stribling, J., Morris, R., Kaashoek, M.F.: Bandwidth-efficient management of DHT routing tables. In: Proceedings of the 2nd USENIX Symposium on Networked Systems Design and Implementation (NSDI '05), Boston, Massachusetts (May 2005)

[94] Newman, M.: The structure and function of complex networks (2003)

[95] Albert, R., Barabási, A.: Statistical mechanics of complex networks

[96] Galuba, W., Aberer, K.: Generic emergent overlays in arbitrary peer identifier spaces. In: 2nd International Workshop on Self-Organizing Systems (IWSOS 2007). Volume 4725. (2007) 88–102

[97] Rhea, S., Chun, B., Kubiatowicz, J., Shenker, S.: Fixing the embarrassing slowness of opendht on planetlab (2005)

[98] Freedman, M.J., Lakshminarayanan, K., Rhea, S., Stoica, I.: Non-transitive connectivity and dhts. In: WORLDS'05, Berkeley, CA, USA, USENIX Association (2005) 10–10

[99] Dumitriu, D., Knightly, E., Kuzmanovic, A., Stoica, I., Zwaenepoel, W.: Denial-of-service resilience in peer-to-peer file sharing systems. SIGMETRICS Perform. Eval. Rev. **33**(1) (2005) 38–49

[100] Rhea, S.C., Geels, D., Roscoe, T., Kubiatowicz, J.: Handling churn in a DHT. In: USENIX, USENIX (2004) 127–140

[101] Freedman, M.J., Freudenthal, E., Mazières, D.: Democratizing content publication with coral. In: NSDI'04, USENIX (2004) 239–252

[102] Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F., Morris, R.: Designing a DHT for low latency and high throughput. In: NSDI'04, USENIX (2004) 85–98

[103] Li, Stribling, Gil, Morris, Kaashoek: Comparing the performance of distributed hash tables under churn. In: IPTPS, LNCS. Volume 3. (2004)

[104] Artigas, M.S., Lopez, P.G., Skarmeta, A.F.G.: A novel methodology for constructing secure multipath overlays. IEEE Internet Computing **09**(6) (2005) 50–57

[105] Andersen, D.G., Snoeren, A.C., Balakrishnan, H.: Best-path vs. multi-path overlay routing. In: IMC '03, New York, NY, USA, ACM (2003) 91–100

[106] Klemm, F., Le Boudec, J.Y., Kostic, D., Aberer, K.: Improving the Throughput of Distributed Hash Tables Using Congestion-Aware Routing. In: IPTPS. (2007)

[107] Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The eigentrust algorithm for reputation management in P2P networks. In: WWW. (2003) 640–651

[108] Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: CIKM. (2001) 310–317

[109] Xiong, L., Liu, L.: Peertrust: Supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans. Knowl. Data Eng **16**(7) (2004) 843–857

[110] Di Caro, G., Dorigo, M.: AntNet: Distributed stigmergetic control for communications networks. Journal of AI Research **9**(2) (1998) 317–365

[111] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of dht routing geometry on resilience and proximity. In: SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, ACM Press (2003) 381–394

[112] Dabek, F., Cox, R., Kaashoek, M.F., Morris, R.: Vivaldi: a decentralized network coordinate system. In: SIGCOMM'04, ACM (2004) 15–26

[113] Wong, B., Slivkins, A., Sirer, E.: Meridian: A lightweight network location service without virtual coordinates. In: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, ACM (2005) 85–96

[114] Galuba, W., Aberer, K., Despotovic, Z., Kellerer, W.: Authentication-free fault-tolerant peer-to-peer service provisioning. In: DBISP2P 2007, Springer (2007)

[115] Bianchi, S., Serbu, S., Felber, P., Kropf, P.: Adaptive load balancing for DHT lookups. In: ICCCN. (2006) 411–418

[116] Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In Almeida, J.M., Almeida, V.A.F., Barford, P., eds.: IMC'06, ACM (2006) 189–202

[117] Galuba, W., Aberer, K., Despotovic, Z., Kellerer, W.: Authentication-free fault-tolerant peer-to-peer service provisioning. In: Databases, Information Systems and Peer-to-Peer Computing, DBISP2P 2007, Springer (2007)

[118] Galuba, W., Aberer, K., Despotovic, Z., Kellerer, W.: Self-organized fault-tolerant routing in peer-to-peer overlays. In: Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems(SASO'09). (2009)

[119] Papadimitratos, P., Haas, Z.: Secure data communication in mobile ad hoc networks. Selected Areas in Communications, IEEE Journal on **24**(2) (2006) 343–356

[120] Papadimitratos, P., Haas, Z., Papadimitratos, P., Haas, Z.J.: Secure rotuing for mobile ad hoc networks. In: in SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002. (2002) 27–31

[121] Hu, Y., Perrig, A., Johnson, D.: Ariadne: A secure on-demand routing protocol for ad hoc networks. Wireless Networks **11**(1) (2005)

[122] Acs, G., Buttyan, L., Vajda, I.: Provably secure on-demand source routing in mobile ad hoc networks. IEEE TMC **5**(11) (2006) 1533–1546

[123] Papadimitratos, P., Haas, Z.: Secure Link State Routing for Mobile Ad Hoc Networks. In: Proceedings of the IEEE Workshop on Security and Assurance in Ad Hoc Networks. (2003)

[124] Papadimitratos, P., Haas, Z., Hubaux, J.: How to Specify and How to Prove Correctness of Secure Routing Protocols for MANET. In: IEEE BROADNETS'06. (2006)

[125] Sanzgiri, K., Dahill, B., Levine, B., Shields, C., Belding-Royer, E.: A Secure Routing Protocol for Ad-hoc Networks. In: IEEE ICNP'02. (2002)

[126] Zapata, M.: Secure ad hoc on-demand distance vector routing. ACM SIGMOBILE Mobile Computing and Communications Review **6**(3) (2002) 106–107

[127] Perkins, C., Belding-Royer, E., Das, S., et al.: Ad hoc on-demand distance vector (AODV) routing. RFC 3561 (2003)

[128] Hu, Y., Johnson, D., Perrig, A.: SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks. Ad Hoc Networks **1**(1) (2003) 175–192

[129] Papadimitratos, P., Haas, Z.: Secure message transmission in mobile ad hoc networks. Ad Hoc Networks **1**(1) (2003) 193–209

[130] Eriksson, J., Faloutsos, M., Krishnamurthy, S.: Routing amid colluding attackers. In: IEEE ICNP'07. (2007)

[131] Awerbuch, B., Curtmola, R., Holmer, D., Nita-Rotaru, C., Rubens, H.: ODSBR: An on-demand secure Byzantine resilient routing protocol for wireless ad hoc networks. ACM Transactions on Information and System Security (TISSEC) **10**(4) (2008) 6

[132] Marwaha, S., Tham, C., Srinivasan, D.: A novel routing protocol using mobile agents and reactive route discovery for ad hoc wireless networks. In: ICON'02. (2002)

[133] Hussein, O., Saadawi, T.: Ant routing algorithm for mobile ad-hoc networks (ARAMA). In: IEEE IPCCC'03. (2003) 281–290

[134] Perrig, A., Canetti, R., Tygar, J., Song, D.: The TESLA broadcast authentication protocol. RSA CryptoBytes **5**(2) (2002) 2–13

[135] Papadimitratos, P., Poturalski, M., Schaller, P., Lafourcade, P., Basin, D., Capkun, S., Hubaux, J.: Secure neighborhood discovery: A fundamental element for mobile ad hoc networking. Communications Magazine, IEEE **46**(2) (2008) 132–139

[136] Hu, Y., Perrig, A., Johnson, D.: Rushing attacks and defense in wireless ad hoc network routing protocols. In: ACM WiSe'03. (2003)

[137] Eriksson, J., Krishnamurthy, S., Faloutsos, M.: Truelink: A practical countermeasure to the wormhole attack in wireless networks. In: IEEE ICNP'06. (2006)

[138] Sundaresan, K., Anantharaman, V., Hsieh, H., Sivakumar, A.: ATP: A reliable transport protocol for ad hoc networks. IEEE TMC **4**(6) (2005) 588–603

[139] Galuba, W., Papadimitratos, P., Poturalski, M., Aberer, K., Despotovic, Z., Kellerer, W.: Castor: Scalable Secure Routing for Ad-hoc Networks. In: 29th Conference on Computer Communications (INFOCOM'10). (2010)

[140] Galuba, W., Papadimitratos, P., Poturalski, M., Aberer, K., Despotovic, Z., Kellerer, W.: More on castor: the scalable secure routing protocol for ad-hoc networks. Technical Report LSIR-REPORT-2009-002, EPFL (2009)

[141] Java, A., Song, X., Finin, T., Tseng, B.: Why we twitter: understanding microblogging usage and communities. In: WebKDD/SNA-KDD '07: Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis. (2007)

[142] Banerjee, N., Chakraborty, D., Dasgupta, K., Mittal, S., Joshi, A., Nagar, S., Rai, A., Madan, S.: User interests in social media sites: an exploration with micro-blogs. In: Proceeding of the 18th ACM conference on Information and knowledge management, ACM (2009) 1823–1826

[143] danah boyd, Golder, S., Lotan, G.: Tweet, Tweet, Retweet: Conversational Aspects of Retweeting on Twitter. In: Proceedings of HICSS-42. (2010)

[144] Cha, M., Mislove, A., Adams, B., Gummadi, K.P.: Characterizing social cascades in flickr. In: WOSN '08: Proceedings of the first workshop on Online social networks. (2008)

[145] Leskovec, J., McGlohon, M., Faloutsos, C., Glance, N., Hurst, M.: Cascading behavior in large blog graphs (2007)

[146] Wu, F., Huberman, B., Adamic, L., Tyler, J.: Information flow in social groups. Physica A: Statistical Mechanics and its Applications **337**(1-2) (2004) 327–335

[147] Szabo, G., Huberman, B.: Predicting the popularity of online content. Communications of the ACM **53**(8) (2010) 80–88

[148] Lerman, K., Galstyan, A.: Analysis of social voting patterns on digg. In: Proceedings of the first workshop on Online social networks, ACM (2008) 7–12

[149] Jackson, M., Yariv, L.: Diffusion on social networks. In: Economie Publique. Volume 16. (2005) 3–16

[150] Adar, E., Adamic, L.: Tracking information epidemics in blogspace. In: IEEE/ACM International Conference on Web Intelligence. (2005) 207–214

[151] Nekovee, M., Moreno, Y., Bianconi, G., Marsili, M.: Theory of rumour spreading in complex social networks. In: Physica A: Statistical Mechanics and its Applications. Volume 374. (2007) 457–470

[152] Granovetter, M.: Threshold models of collective behavior. American journal of sociology **83**(6) (1978) 1420

[153] Goldenberg, J., Libai, B., Muller, E.: Talk of the network: A complex systems look at the underlying process of word-of-mouth. Marketing Letters **12**(3) (2001) 211–223

[154] Kempe, D., Kleinberg, J., Tardos, É.: Maximizing the spread of influence through a social network. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM New York, NY, USA (2003) 137–146

[155] Krishnamurthy, B., Gill, P., Arlitt, M.: A few chirps about twitter. In: Proceedings of the first workshop on Online social networks, ACM (2008) 19–24

[156] Huberman, B., Romero, D., Wu, F.: Social networks that matter: Twitter under the microscope. First Monday **14**(1) (2009) 8

[157] Kwak, H., Lee, C., Park, H., Moon, S.: What is Twitter, a social network or a news media? In: Proceedings of the 19th international conference on World wide web, ACM (2010) 591–600

[158] Galuba, W., Aberer, K., Chakraborty, D., Despotovic, Z., Kellerer, W.: Outtweeting the Twitterers - Predicting Information Cascades in Microblogs. In: 3rd Workshop on Online Social Networks (WOSN'10). (2010)

[159] Sirivianos, M., Park, J., Chen, R., Yang, X.: Free-riding in BitTorrent Networks with the Large View Exploit. In: Proc. of IPTPS. (2007)

[160] Locher, T., Moor, P., Schmid, S., Wattenhofer, R.: Free Riding in BitTorrent is Cheap. HotNets **300** (2006) 400

[161] Jun, S., Ahamad, M.: Incentives in BitTorrent induce free riding. In: Proc. of the 2005 ACM SIGCOMM workshop on Economics of P2P systems, ACM New York, NY, USA (2005)

[162] Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., Venkataramani, A.: Do incentives build robustness in BitTorrent. In: NSDI. (2007)

[163] Biddle, P., England, P., Peinado, M., Willman, B.: The Darknet and the Future of Content Distribution. In: Proceedings of the 2002 ACM Workshop on Digital Rights Management. (2002)

[164] Rogers, M., Bhatti, S.: How to disappear completely: A survey of private peer-to-peer networks. In: SPACE 2007. (2007)

[165] Li, J., Dabek, F.: F2F: Reliable storage in open networks. In: IPTPS'06. (2006)

[166] Leskovec, J., Horvitz, E.: Worldwide Buzz: Planetary-Scale Views on an Instant-Messaging Network. In: WWW. (2008)

[167] Sandberg, O.: Distributed Routing in Small-World Networks. In: Proc. of the Eighth Workshop on Algorithm Engineering and Experiments, Society for Industrial Mathematics (2006)

[168] Pouwelse, J., Garbacki, P., Wang, J., Bakker, A., Yang, J., Iosup, A., Epema, D., Reinders, M., van Steen, M., Sips, H.: Tribler: a social-based peer-to-peer system. Concurrency and Computation **20**(2) (2008) 127

[169] Kulbak, Y., Bickson, D.: The eMule Protocol Specification. eMule project, http://sourceforge. net

[170] Locher, T., Schmid, S., Wattenhofer, R.: Rescuing Tit-for-Tat with Source Coding. In: P2P'07. (2007)

[171] Piatek, M., Isdal, T., Krishnamurthy, A., Anderson, T.: One hop reputations for peer to peer file sharing workloads. In: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation, USENIX Association (2008) 1–14

[172] Bethencourt, J., Low, W., Simmons, I., Williamson, M.: Establishing darknet connections: an evaluation of usability and security. In: Proceedings of the 3rd symposium on Usable privacy and security, ACM Press New York, NY, USA (2007) 145–146

[173] Isdal, T., Piatek, M., Krishnamurthy, A., Anderson, T.: Friend-to-friend data sharing with oneswarm. Technical report, UW-CSE (2009)

[174] Dischinger, M., Haeberlen, A., Gummadi, K., Saroiu, S.: Characterizing residential broadband networks. In: IMC'07. (2007)

[175] Ganesan, P., Seshadri, M.: On Cooperative Content Distribution and the Price of Barter. In: ICDCS. (2005)

[176] Alix L.H. Chow, L.G., Misra, V.: Improving BitTorrent: A Simple Approach. In: Proc. of IPTPS. (2007)

[177] Galuba, W., Aberer, K., Despotovic, Z., Kellerer, W.: Leveraging social networks for increased Bittorrent robustness. In: 7th Annual IEEE Consumer Communications and Networking Conference (CCNC'10). (2010)

[178] Galuba, W.: Friend-to-friend computing: Building the social web at the internet edges. Technical Report LSIR-REPORT-2009-003, EPFL (2009)

[179] Naicken, S., Livingston, B., Basu, A., Rodhetbhai, S., Wakeman, I., Chalmers, D.: The state of peer-to-peer simulators and simulations. SIGCOMM Comput. Commun. Rev. **37**(2) (2007) 95–98

[180] Vahdat, A., Yocum, K., Walsh, K., Mahadevan, P., Kostić, D., Chase, J., Becker, D.: Scalability and accuracy in a large-scale network emulator. SIGOPS Oper. Syst. Rev. **36**(SI) (2002) 271–284

[181] Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. **33**(3) (2003) 3–12

[182] : Jist/swans. http://jist.ece.cornell.edu/ (Mar 2008)

[183] Klemm, F., Le Boudec, J.Y., Aberer, K.: Congestion Control for Distributed Hash Tables. In: The 5th IEEE International Symposium on Network Computing and Applications (IEEE NCA06). (2006)

[184] Dabek, F., Li, J., Sit, E., Robertson, J., Kaashoek, M.F., Morris, R.: Designing a DHT for low latency and high throughput. In: NSDI, USENIX (2004) 85–98

[185] Matsumoto, M., Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation (TOMACS) **8**(1) (1998) 3–30

[186] Naicken, S., Basu, A.and Livingston, B., Rodhetbhai, S.: A survey of peer-to-peer network simulators. In: Proceedings of The Seventh Annual Postgraduate Symposium. (2006)

[187] : Narses. http://sourceforge.net/projects/narses (Mar 2008)

[188] : Overlay weaver. http://overlayweaver.sourceforge.net/ (Mar 2008)

[189] : Planetsim. http://www.planetsim.net/ (Mar 2008)

[190] : Peersim. http://peersim.sourceforge.net/ (Mar 2008)

[191] : P2psim. http://pdos.csail.mit.edu/p2psim (Mar 2008)

[192] : Neurogrid. http://www.neurogrid.net/ (Mar 2008)

[193] Yang, W., Abu-Ghazaleh, N.: Gps: a general peer-to-peer simulator and its use for modeling bittorrent. 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (2005) 425–432

[194] Killian, C.E., Anderson, J.W., Braud, R., Jhala, R., Vahdat, A.M.: Mace: language support for building distributed systems. SIGPLAN Not. **42**(6) (2007) 179–188

[195] Loo, B.T., Condie, T., Hellerstein, J.M., Maniatis, P., Roscoe, T., Stoica, I.: Implementing declarative overlays. SIGOPS Oper. Syst. Rev. **39**(5) (2005) 75–90

[196] Arad, C., Kafray, O., Ghodsi, A., Haridi, S.: GODS: Global observatory for distributed systems. Technical report, Swedish Institute of Computer Science (2007)

[197] Song, H.: The MicroGrid: A scientific tool for modeling Computational Grids. Scientific Programming **8**(3) (2000) 127–141

[198] Urban, P., Defago, X., Schiper, A.: Neko: a single environment to simulate and prototype distributedalgorithms. In: Information Networking, 2001. Proceedings. 15th International Conference on. (2001) 503–511

[199] Quinson, M.: GRAS: A research & development framework for grid and P2P infrastructures. In: International Conference on Parallel and Distributed Computing and Systems. (2006)

[200] Casanova, H., Legrand, A., Quinson, M.: SimGrid: a Generic Framework for Large-Scale Distributed Experiments. In: 10th IEEE International Conference on Computer Modeling and Simulation. (March 2008)

[201] White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., Joglekar, A.: An integrated experimental environment for distributed systems and networks. ACM SIGOPS Operating Systems Review **36**(SI) (2002) 255–270

[202] : ns2 network simulator. http://www.isi.edu/nsnam/ns/ (Mar 2008)

[203] Galuba, W., Aberer, K., Despotovic, Z., Kellerer, W.: Protopeer: From simulation to live deployment in one step. Peer-to-Peer Computing , 2008. P2P '08. Eighth International Conference on (2008) 191–192

[204] Galuba, W., Aberer, K., Despotovic, Z., Kellerer, W.: ProtoPeer: A P2P Toolkit Bridging the Gap Between Simulation and Live Deployment. In: 2nd International Conference on Simulation Tools and Techniques (SIMUTools'09). (2009)

Appendix A

ProtoPeer: a Distributed Systems Toolkit Bridging the Gap Between Simulation and Live Deployment

Throughout the thesis, we have proposed several different protocols for various types of distributed systems. To aid the systematic repeatable protocol evaluation both in simulation and in live Internet deployments, we have developed a distributed systems development toolkit, ProtoPeer, which we present in this appendix.

A.1. Introduction

The three important tools used by the distributed systems research community are the analytical models, simulations and experiments on the actual systems. Large-scale distributed systems are complex and accurately modeling them analytically is not an easy task. Most of the time, the first iteration of an analytical model is not tractable and the model needs to be successively simplified to produce useful insights. This leads to models that describe the system at a very coarse-grained level and with many uniformity assumptions. Despite this, the analytical models are in most cases accurate, scale to an arbitrarily large system size and are an excellent tool for early feasibility assessment of a distributed systems solution. However, for a more complete and accurate evaluation under a wider range of conditions the researchers commonly turn to simulation or system evaluation on the actual networks.

Simulations cannot be scaled to an arbitrarily large size as the analytical models can, but allow for exact implementation of the message passing protocols. The protocols can then be evaluated under a wide range of conditions without being limited to the highly uniform cases used in the analytical models. With all their advantages, simulators often make many simplifying assumptions about the underlying network model, which might affect the predictions obtained from the simulation in non-trivial ways.

Usually, the simulators are purpose-built for specific applications or classes of applications. Few simulators are designed as more general tools for system building and evaluation. What is more, even though simulation is a widely used evaluation technique there is almost no simulator code sharing among the researchers and little standardization of the common practices. Naicken et. al [179] examine 287 papers on peer-to-peer systems. Out of the 141 papers that use some form of simulation 114 either use a custom-made simulator or do not specify what software was used. The other 27 papers use publicly available simulators, however these papers are either authored by researchers who developed the simulator or by researchers from the university in which the simulator was developed. The limited evaluation tool reuse motivated us to invest effort in sharing our framework, ProtoPeer, with the rest of the community. We have also made ProtoPeer modular such that it can be easily extended and the modules can be shared among the system builders. We further define the niche ProtoPeer fills and compare our framework to the others in §A.6.

While simulators are a popular tool, the most accurate system evaluation can only be achieved with a system implementation running in the real network. Switching from the simulation to the actual implementation often requires a considerable development effort. Network I/O implementation, measurement instrumentation, deployment automation and distributed debugging are among the most time consuming tasks that developers face. Moreover, large parts of the application code existing in the simulator are not reused in the actual system. Bridging the gap between the simulation and the actual system deployment was the primary motivation for developing ProtoPeer.

In ProtoPeer the developer can switch between the simulation of a P2P system to its deployment on the actual network without changing a single line of code. This dramatically speeds up the implement-evaluate-reimplement cycle. Most of the major bugs and performance problems are caught early on during the simulation while the more time-consuming live deployment is used for the accurate evaluation of the final system on testbeds such as ModelNet [180] or PlanetLab [181].

ProtoPeer has an API for building arbitrary message passing systems not only the peer-to-peer systems. Most of the common tasks such as network I/O, message serialization and message queuing are handled by ProtoPeer. The user only needs to focus on implementing the message passing logic of the application without worrying about the underlying details. If need arises, the ProtoPeer API allows users to plug in their own implementations of most of the parts of the system. This can be for example used to implement a specialized message queuing discipline or to implement message passing over transports other than the default TCP or UDP.

Measurements are an important part of any system evaluation. ProtoPeer has tools and APIs covering most of the measurement pipeline: from measurement instrumentation through aggregation and computation of the basic statistics to plotting the measurements at the end. ProtoPeer also implements a general event injection mechanism, which is particularly useful for evaluating the system under various failure scenarios and modeling churn (i.e., peer departures and arrivals).

A.2. Architecture

A.2.1. Message passing & timers

In ProtoPeer the system is composed of a set of peers that communicate with one another by passing messages. Each application[1] defines its set of messages and message handlers. An application typically also defines a set of timers and handlers for the timer expiration events. All the application logic in ProtoPeer is called from within the timer and message handlers.

A.2.2. Networking & time abstraction

One of the main goals of ProtoPeer is to be able to switch between simulation and live network deployment without changing any of the application's code. The key architectural feature that enables this are the abstract time and networking APIs[2]. The APIs allow for only a small number of basic operations: creation of timers for execution scheduling and creation of network interfaces for sending and receiving messages. These simple APIs serve as the key building block for the rest of the ProtoPeer and form the "waist" of the framework's hourglass architecture (Fig. A.1).

When switching from the simulated run to the live run the simulated time and networking implementations are simply swapped with the implementations using real timers and TCP or UDP networking (see §A.4.1). The ProtoPeer user can also provide alternative time and networking implementations, using, for example, some other transport protocols or different message serialization. The new implementations can be plugged in without any changes to the application code using them. We elaborate further on network modeling in §A.2.4.

A.2.3. Event-driven execution

Execution in ProtoPeer progresses by calling event handlers in response to time and networking events. For example, when a timer expires an appropriate handler is called for that event. The handler might send a message or schedule other timers, which subsequently trigger other handler calls and so on.

During simulation the events are stored in a single system-wide queue. The events are ordered according to their scheduled time of execution (the time is virtual). When a handler call finishes the next closest future event is dequeued and its corresponding handler is executed.

[1] by "application" we will understand a distributed application implemented using ProtoPeer, there can be several applications running simultaneously in the system

[2] The reader is referred to the on-line ProtoPeer tutorial at `http://protopeer.epfl.ch/wiki/IntroTutorial` for numerous code examples. We have omitted them in the thesis.

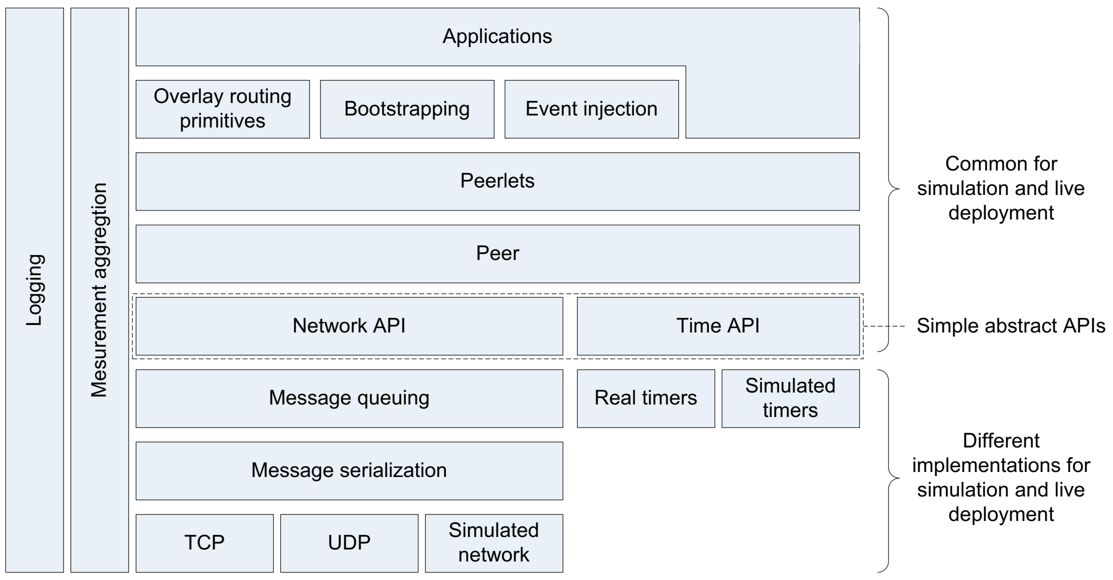


Figure A.1. The ProtoPeer architecture. The time API and the networking API consist of only a few basic abstractions and methods and form the narrow "waist" of the ProtoPeer's hourglass architecture. The upper part of the hourglass are all the components that use the time and networking APIs. The peer provides the runtime context for the peerlets (§A.2.6) in which the various parts of the peer's functionality are encapsulated, e.g., bootstrapping logic, overlay message routing, event injection (§A.3.2) or the application-specific logic. The lower part of the hourglass are the concrete implementations of the abstract time and networking APIs, e.g., message passing over TCP or virtual timers for scheduling events during simulation. Switching from the simulated system to the actual system is as simple as switching from one time & networking implementation to another.

During live deployment handlers are asynchronously called as the various networking and timer events happen. The ProtoPeer's event-driven architecture does not forbid concurrent execution of two handlers in two different threads. The implementations of time and networking have complete freedom in allocating the different handler calls to the threads, which gives great flexibility in performance optimization. In the current version of ProtoPeer both time and networking implementations use thread pools with configurable size. We discuss the advantages of this approach in §A.4.1.

A.2.4. Network modeling

As mentioned in §A.2.2 switching between the simulation and the live run is as easy as swapping one networking implementation for the other. During simulation the network needs to subject the messages to realistic delay and loss. Loss and delay modelling are encapsulated in the `NetworkModel` interface in ProtoPeer. Users can provide their own implementations of that interface. There are several implementations already available, including simple uniformly distributed delay model, the Euclidean model (i.e., delay between nodes proportional to their distance in the Euclidean space) or the delay matrix model into which arbitrary delay matrices can be loaded (e.g., based on the King dataset[3]).

Messages in ProtoPeer can be of arbitrary size after serialization. The delay to which the message is subjected should be a function of its size and the available bandwidth. For simulating bandwidth allocation we are currently developing a MaxMin flow-based model. In that model each peer has a limited incoming and outgoing bandwidth. This model is especially useful for simulating bandwidth-bound applications such as BitTorrent, while the other simpler and faster models mentioned in the previous paragraph can be used for delay-bound applications such as DHTs.

The flexibility of ProtoPeer's networking APIs allowed us to implement a separate network model for mobile ad-hoc networks (MANETs) using the JiST/SWANS framework [182]. The model was simply plugged into the simulator and the same message passing protocols that run in peer-to-peer systems could then be evaluated in MANETs. This approach allowed us to iterate the design of the Forward Feedback Protocol (§4) and develop Castor, its MANET variant (§5).

A.2.5. Overlay modeling

Research on peer-to-peer systems has produced many ways of constructing and maintaining overlays. Most overlays assign identifiers to the peers from some ID space. Each ID space typically has a distance metric associated with it. For example, in Chord [36] the ID space is the unit ring (0,1] and the metric is the distance between the IDs on the ring, while in Kademlia[37] peers take IDs from the set of 160-bit integers, the distance is measured as the numerical value of the bitwise XOR between two IDs. In fact, there exist overlays construction protocols for arbitrary identifier spaces and distance metrics[96]. To support the wide range of overlays, ProtoPeer defines an abstract `PeerIdentifier` that is used throughout the system. The different overlay implementations override it with concrete implementations of their ID space and the distance metric.

Most of the peer-to-peer systems define some form of logical links between peers that together form the overlay topology. This concept is so fundamental that it has been added at the core of ProtoPeer. Each peer has its neighbor set and exposes it in a uniform way to all applications. Each neighbor is stored as a pair of the neighbor's peer ID and the neighbor's network address. The neighbor set appears in many overlay implementations and is sometimes referred to as the routing table, finger table etc. The neighbor set is used not only by the overlays but also by the applications running on top of them such as DHTs, which is another reason for making access to the peer's neighbors uniform throughout the system.

[3] http://pdos.csail.mit.edu/p2psim/kingdata/

A.2.6. Peerlets

A peer in the peer-to-peer system typically implements more than one piece of the message passing functionality. For example, a peer might need one protocol for contacting the bootstrap server and getting the initial neighbors, another protocol for maintaining the overlay during churn and yet another for DHT key replication. In ProtoPeer the message passing logic and state of each of the protocols is encapsulated in components called *peerlets*. Peers are constructed by putting several peerlets together. The peerlets can also be removed or added at runtime.

The peerlets, just as the applets or servlets, have the familiar init-start-stop lifecycle. The peer provides the execution context for all of the peerlet instances it contains. The peerlets can discover one another within that context and use one another's functionality. The peerlets have access to the peer's network interface through which they send and receive messages. Peerlets can also arbitrarily modify the peer's neighbor set.

The peerlet-based approach has all the advantages of any other modular design. Firstly, the message passing functionality is conveniently encapsulated in building blocks with well defined behavior. The blocks can be composed to achieve the desired peer functionality. Certain functionality can be easily enabled or disabled depending on the context (e.g., debug mode vs. evaluation mode). Secondly, peerlets can be reused across applications. Peerlets can export well defined interfaces e.g., a DHT interface, which can have several implementations that can be easily swapped one for another. Lastly, peerlets can be unit tested either in isolation or with other peerlets as mock objects.

A.2.7. Queuing

Message queues are an essential component of many peer-to-peer system designs. Queues buffer the messages during the transient periods when the rate of asynchronously arriving messages at the peer exceeds its capacity to process them. Very often queues are not explicitly implemented and the applications implicitly rely on operating system's socket buffers for message buffering. Even if the queues are implemented they are typically unbounded. This is a valid assumption for the applications with low message rates. However, queuing has to be more carefully designed in systems optimizing for high throughput or implementing congestion control schemes (for example [183, 184]).

In ProtoPeer all the peers have incoming and outgoing message queues (Fig. A.2). Applications may add their own queue implementations. At runtime the applications have access to the queue state and can for example use it for congestion signaling. Queues are a versatile tool and can be used for many purposes: injection (§A.3.2) by dropping, mutating or delaying messages.

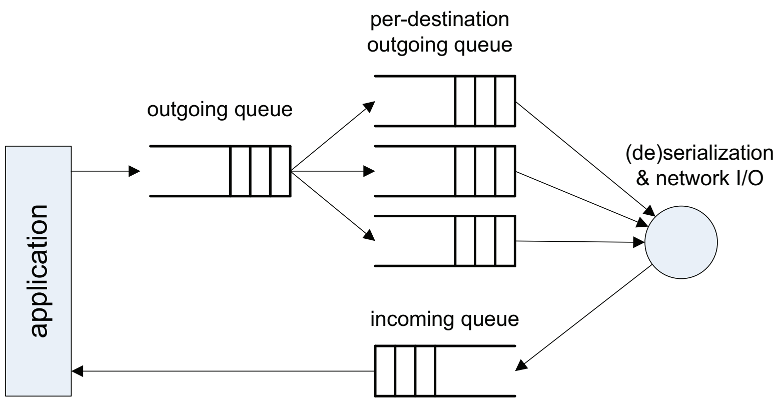

per-destination outgoing queue
outgoing queue
(de)serialization & network I/O
application
incoming queue

Figure A.2. Message queueing in ProtoPeer. When an application sends a message it first passes through the main outgoing queue and then is placed on the appropriate per-destination outgoing queue. The networking layer picks the messages from the outgoing queue, serializes them and sends the raw binary data over the network. When data is received, the deserialized message is put on the incoming queue for the application to pick up. Each of the queues can be replaced by application-specific implementations. Queues can consist of from several queues linked together in various ways. Basic implementations of mux/demux queues, queue chains etc. are available.

A.3. Tools

So far we have covered the basic architectural building blocks for constructing a peer-to-peer system. We now turn to facilities available in ProtoPeer for evaluating the peer-to-peer systems once they are built.

A.3.1. Measurement infrastructure

Obtaining reliable and accurate measurements is an important, if not the most important part of any peer-to-peer system evaluation. Measurements need to be instrumented in the application code, logged, aggregated from all the peers in the system, analyzed and optionally plotted. ProtoPeer's measurement infrastructure helps with all of these tasks. The measurements are instrumented by doing calls to the measurement API in the appropriate places in the application code. While the system is running the measurements can either be sent regularly to the measurement server for live monitoring of the network or they can be dumped to a local file at each peer and aggregated later.

While the measurements are accumulating the system computes the basic statistics on-the-fly: average, sum, variance etc. This allows for extremely compact representation of the measurements. Optionally all logged values can be kept which later on permits the computation of other statistics such as percentiles. The statistics can be computed at various aggregation levels: per peer, per time window and per measurement tag. Measurement tags are objects used to "mark" the values that are reported to the measurement logger. The user can request

```
//count messages per class
mlog.log("count", incomingMessage.getClass(),1);
//log the measured round-trip-times
mlog.log("rtt",rttValue);
//log the traffic flow for every link (in bytes)
mlog.log("outflow", sourceAddress, destinationAddress,
         outgoingMessage.getSize());

//get total number of DHTLookup messages
mlog.getAggregate("count", DHTLookup.class).getSum()
//get the 95th percentile of the RTTs
mlog.getAggregate("rtt").getPercentile(95);
//get the total number of bytes TXed from one IP to another
mlog.getAggregate("outflow",new NetworkAddress("62.35.31.65:3392"),
         new NetworkAddress("132.35.136.25:6342")).getSum();
```

Figure A.3. Measurement API example. Measurement logging is done by calling `log()` on the `MeasuremenLogger` instance. The function takes as arguments an arbitrary number of tags and the measured floating point value. A tag can be any Java object as long as its `hash()` and `equals()` methods are well defined, which makes the tagging-based measurement a multi-purpose tool. The same tags used for logging are then used to obtain aggregates. Aggregates can also be computed for specific time windows, specific peers or system-wide. There are tools for fetching and merging measurement logs from all the peers as well as tools for log browsing and plotting.

an aggregate performed over all the measurements that match a certain tag or a set of tags. Tags can be any object, in particular they can identify an individual peer, a link between peers, the message class or they are simply a string describing the nature of the measured value. We found that in practice the simple tagging-based measurement infrastructure covers the vast majority of needs (Fig. A.3). Alternative solutions were necessary only in very obscure cases where complex objects had to be logged as the "measured values", which can be done using the existing logging frameworks [4]

ProtoPeer has several tools to support the measurement post-processing phase. Measurement log files coming from the different peers can be merged into a single system-wide log which can then analyzed separately. The access to the log content is programmatic so the users have complete freedom in outputting the processed log information in any format they desire. There is a basic tool that allows to browse the contents of the log files and plot measurements for the various aggregates types and tags.

Measurement logging can be disabled to minimize its impact on the system performance and for evaluating the "production" version of the system.

A.3.2. Event injection & scenarios

While evaluating the peer-to-peer system there is frequently a need to test the system's response to various, often exogenous events, e.g., peer arrivals and departures (i.e., churn), user actions

[4] http://logging.apache.org/log4j/

```
#set up the churn sequence
4        14.3     Peer.start()
3        15.1     Peer.stop()
1        36.9     Peer.start()
4        44.4     Peer.stop()

#inject a failure at 150s on 10 peers
0-9      150.0    Peer.Router.setDropMessages(true)
```

Figure A.4. An example scenario file. Each scenario file consists of three columns. The first one specifies the peer index (or a range of indices) that uniquely identify the affected peer. The second column is the number of seconds since the beginning of the simulation when the event should be injected. The last column indicates the method to be called. Churn can be simply defined as a sequence of calls to the peer's start and stop methods. Calls can be made to any method of the peer or its peerlets. Users can define their own methods and call them in the scenario files, which makes event injection a multipurpose tool.

or, more commonly, failures. ProtoPeer provides a simple but general mechanism for event injection. The events are specified in triples consisting of time, the set of unique peer indices to be affected and the method to call. Despite its simplicity this way of describing events is expressive enough to cover most of the common use cases.

A set of events defines a *scenario*. Peers load the scenarios on startup and execute the events specified in them. Each scenario can be kept in a separate file (Fig. A.4). For example, one scenario file can contain the precise churn model for the system specifying when the peers should go online or offline, while another scenario file can define the failures injected at the different moments in time. The scenario files can be generated, merged and filtered in various ways using the common text processing utilities. Scenarios are an important tool for systematizing the evaluation process and ensuring high experiment repeatability.

A.3.3. Unified randomness source

Another facility for ensuring repeatability in ProtoPeer is the unified source of randomness in the system. All random numbers are drawn via single system component managing the random number generators. The generators are seeded at the beginning. The same seed leads to the same sequence of events, which is particularly important during debugging. Naturally, randomness can only be completely controlled during simulation, however, if used correctly the unified randomness source also improves the repeatability of live runs. ProtoPeer uses the Mersenne twister [185] random number generator. Users can plug in their own generators if needed.

ProtoPeer manages several random generators at a time, each for a different purpose. This allows to, for example, keep the randomly generated overlay topology the same while changing the random sequence of message delays.

A.4. Implementation

ProtoPeer is developed in Java, which has been chosen for its popularity, ease-of-use and availability of libraries. We next cover the key implementation details.

A.4.1. Networking

In most of the peer-to-peer systems, messages arrive at the peers asynchronously. We have implemented both the time and networking event handling using the thread pools. The application puts the incoming messages on the queue, there is a pool of processing threads, whenever one of the threads becomes idle it dequeues the next message and processes it. We expose the queues to the application §A.2.7 so that the application can access their state or replace them with its own custom queue implementations.

Networking is implemented using Apache MINA[5], a high-performance networking framework. Its event-driven design and the use of non-blocking I/O fits well into ProtoPeer. Messages can be sent either over UDP or TCP. The choice which of the two to use can be made by the application on the per-message basis. All the complexity of opening sockets, maintaining them, handling the I/O errors and serializaing/deserializing messages is hidden from the application, which only receives callbacks for successfully sent messages and network exceptions.

A.4.2. Messages

Each message type in ProtoPeer is a separate Java class. The fields of the class correspond to the fields of the message. Messages can contain fields of any type, not only the primitive types, all of which are correctly passed between the peers.

During a live run when a peer sends a message it is serialized into bytes and then deserialized at the receiving peer. During the simulation the messages are efficiently cloned using the standard Java cloning mechanisms. Cloning is necessary to ensure that any subsequent modifications of the message instance at the source after the send call returns do not affect the message instances received at the destinations. Message classes that are guaranteed to be immutable can easily disable cloning. To measure bandwidth consumption during simulation, cloning can be optionally replaced with serialization, naturally, at a performance cost.

The standard Java serialization mechanism is extremely verbose and is primarily designed with persistent storage in mind not for transient data sent over the network. To address this, ProtoPeer uses its own *lightweight serialization* protocol which reduces the bandwidth consumption by a factor of 3 to 10 compared to the standard Java serialization. New message classes do not have to implement any serialization code, lightweight serialization is done through the Java Reflection API. Optionally, if message serialization becomes a performance bottleneck for some messages, they can define their own optimized binary serialization protocol. We are

[5] http://mina.apache.org/

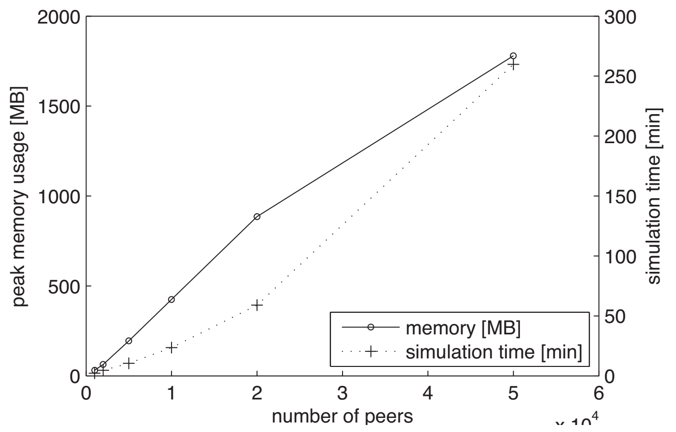


Figure A.5. ProtoPeer simulator scaling. As the number of peers increases, the number of messages that are passed around in the system is increasing at least linearly as each added peer contributes its own lookup workload to the system. There are approximately 20 different measurement instrumentation points, the measurements are aggregated on-the-fly during the simulation. Logging is turned off. The system used was a 4-core Xeon 3.4GHz with 3GB of RAM running 2.6.8 Linux and Sun JDK 1.6.

also considering dynamically injecting the serialization code into the Java classes using the Java bytecode rewriting to avoid the reflection overhead. Efficient serialization is still work in progress and we omit it in the evaluation section.

We have not yet considered the interoperability of ProtoPeer applications with other systems, however, the serialization is well separated from the rest of the framework and custom code can be added for serializing messages into a standard format, e.g., XML-based.

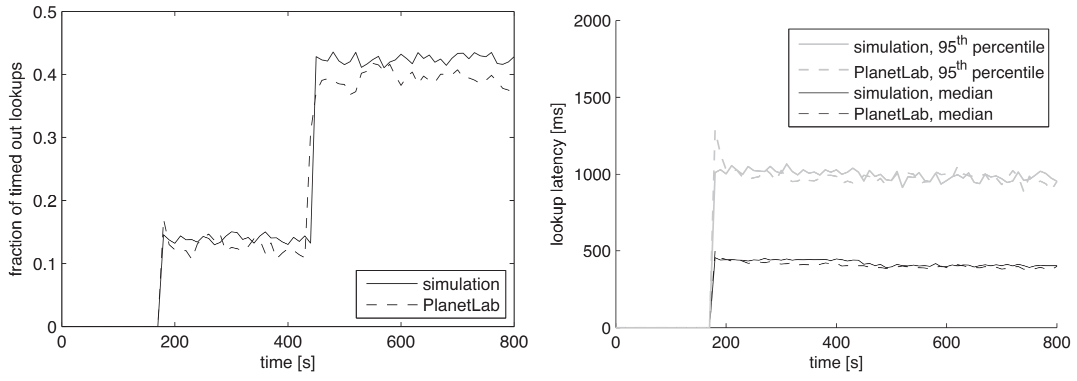
A.5. Performance

The goal of this section is to to evaluate the main performance characteristics of ProtoPeer. The results presented here should be treated as indicative, they are likely to change as the framework's codebase is evolving.

A.5.1. Scalability

We place the peers on a one-dimensional ring and wire them according to the Chord[36] rules. Each peer at 500-1500ms intervals picks a key from the ID space uniformly at random and performs a lookup on that key. The lookup is routed using the Chord algorithm. We simulate 300 seconds of the running system. The number of peers is gradually increased to test the scalability of the system (Fig. A.5).

In our Chord routing test ProtoPeer reaches its scalability limit at approximately 50000 peers, when it hits the memory cap of 1.8GB, the maximum Java heap size the JDK 1.6 would allow us to allocate on a 3GB Linux system. The main memory consumer are the messages



(a) Lookup timeouts (b) Lookup latency

Figure A.6. ProtoPeer simulator accuracy. The Chord routing test was run in a 350 peer PlanetLab deployment and in a 350 peer ProtoPeer simulation using the PlanetLab network model. All messages are sent over UDP. We measured the median and the 95^{th} percentile lookup latency as well as the fraction of lookups that timed out, i.e. when the destination did not respond back to the source. The timeout was set to 1500ms. The peers start issuing lookups at 180s. In addition at 450s we inject a failure (§A.3.2), 50 randomly chosen peers stop forwarding the lookup messages. This leads to an increase in the number of timeouts.

in the simulator's event queue scheduled to be delivered to their destinations. The main CPU bottleneck (approx. 20%) is message cloning when messages are passed from one peer to another. However, for most of the systems implemented in ProtoPeer that we worked with the CPU bottlenecks in the application code are far more common than in the ProtoPeer code.

A.5.2. Accuracy & realism

Simulation realism greatly depends on the network model used (§A.2.4). To validate our simulator's accuracy we have developed the *NetMapper* tool for generating network models that reproduce the exact delay and loss conditions occurring in a live network. We deployed NetMapper on 350 PlanetLab hosts. For 3 hours the hosts were pinging one another at one minute intervals. For each link the message loss and roundtrip latency were measured. Based on the measurements for each link we then generate a log-normally distributed delay model and compute the fixed loss probability based on the fraction of messages that got through on a given link. The delay and loss models are then plugged into ProtoPeer. Different delay and loss scenarios can then simply be generated by changing the random seed. This removes the need for capturing many PlanetLab traces and replaying them in the simulator.

To test the accuracy of ProtoPeer simulation and our delay-loss model we have re-run the same Chord routing test as in §A.5.1. We simulated a 350 peer Chord system and ran the same system live on PlanetLab. Figure A.6 compares the measurements from both runs. The latency predictions from the simulator are very close to the ones obtained from the live network, especially considering the fact that the lookup delay accumulates over many peers and links on the routing path. On the other hand, the message loss is slightly overestimated in simulation, i.e., there are more lookup timeouts. Overall, however, the simulation results stay remarkably close to the ones from PlanetLab, which validates ProtoPeer as an accurate simulation tool. We have consistently observed similar accuracy in the wide range of other systems that we implemented with ProtoPeer. In general, when there is no network congestion on the links or CPU overload at the nodes, the simulation results from our network model are very close to the ones from PlanetLab. Our framework is open-ended and if needed, users can implement more sophisticated models that take congestion into account.

A.6. Related work

A.6.1. Simulators

Naicken et al. [186] survey nine existing peer-to-peer system simulators. The simulators can be broadly classified into flow-level [187], message-level [188, 189, 190] and packet-level [191], in increasing level of simulation detail. ProtoPeer in its current implementation falls into the message-level category, which offers an acceptable level of accuracy for most peer-to-peer applications (§A.2.4).

Despite the lack of packet-level simulation detail, ProtoPeer provides an API for defining the models of the underlying network. Message loss and delay can be accurately simulated. This is in contrast to some of the existing simulators which either ignore the problem of delay and loss simulation or do not provide a way to customize the network model [190, 188, 192].

Most of the existing simulators do not provide any measurement facilities and those that do [191, 188, 190] offer only a default set of measurements, adding new ones requires a considerable development effort. ProtoPeer has a unified API and tools covering most of the measurement pipeline (§A.3.1)S: instrumentation, system-wide log merging and aggregate computation (e.g., mean, sum, percentiles).

A number of the existing simulators have the ability to script events in the system [191, 193]. Event injection is an important system evaluation tool. It can be used, for example, for injecting failures into specific system components or for simulating churn by specifying the peer arrival and departure events. ProtoPeer uses simple (but expressive) scenario files for event injection (§A.3.2), which help the user systematize the evaluation process.

A.6.2. System development tools

ProtoPeer is not only a simulator but also a tool for building systems running in real networks and we need to relate our framework to the existing ones.

Mace [194], a C++ language extension, defines a language for message passing protocol specification. Given the protocol specification Mace generates the C++ code which can then be deployed on the peers. One of the distinctive features of Mace is that it allows for automated protocol verification based on the specifications. Unlike ProtoPeer, Mace code cannot be easily run in a simulator, the more common practice with Mace is to use the emulators such as ModelNet [180], which limits the scalability.

In P2 [195] overlays are defined in a declarative language OverLog. OverLog allows for concise representation of protocols. However, OverLog, being a high-level language, hides most of the low-level implementation details which are delegated to the runtime.

In ProtoPeer we opted for the more traditional event-driven way of protocol specification (as in MACE) instead of the more concise declarative way with a steeper learning curve (as in P2). Unlike the two above frameworks we do not define a new language for specifying protocols, the application programmatically sets up the handlers for the various events occurring in the system (§A.2.2) and the execution progresses by calling these handlers (§A.2.3). The same handler setup is used both during the simulation and live deployment.

Neither P2 nor Mace allow for simulation of the system prior to deployment, which is their main disadvantage in comparison to ProtoPeer.

A.6.3. Testbeds & emulators

ProtoPeer requires the users to develop the application with the ProtoPeer API, there is no trivial way of taking the existing unmodified application and evaluating it with ProtoPeer. However, our framework was not designed for that, many testbeds and emulators have been developed with that goal in mind.

GODS [196], provides tools for automating the evaluation of distributed systems, but is not a library for developing them. Similarly to ProtoPeer, GODS has an infrastructure for instrumenting and aggregating measurements (§A.3.1) and for injecting events into the system (§A.3.2), e.g., peer departures and arrivals or link state changes. The framework also has tools for automating the evaluation and controlling the system lifecycle. GODS uses ModelNet [180] for IP layer emulation. In the MicroGrid [197], the application's networking calls are intercepted and mediated by the framework to emulate the network. MicroGrid uses a local scheduler that starts and stops the application processes according to a predetermined configuration.

Although the testbeds and emulators do not require any modifications to the application code, they have limited scalability since each application process is run separately and requires orders of magnitude more resources than application instances running in a simulator.

A.6.4. Develop once, deploy many times

ProtoPeer is designed from ground up to serve the dual role of a simulator and a tool for developing the live-deployable systems. There are several other frameworks capable of this. Most notably, Neko [198], uses the same message passing model as we do. Similarly to ProtoPeer, the messages can either be serialized and sent over TCP or UDP or passed between the peers in a simulator. However, Neko focuses primarily on the application layer and not the networking layer. The simulator has been designed for the clustered distributed systems communicating over LAN rather than for wide-area system communicating over the Internet, which is the target deployment environment for ProtoPeer.

GRAS [199] together with SimGrid [200] form a development and simulation framework. The framework was designed with Grid systems in mind and thus the focus is on bandwidth- and CPU-limited distributed applications while our ProtoPeer was designed for prototyping the delay-limited message passing systems. We are currently working on implementing a network model for ProtoPeer using the MaxMin bandwidth allocation, which would allow the simulation of bandwidth-limited applications such as BitTorrent.

Emulab (Netbed) [201] was originally designed as an emulator and a framework for network virtualization, but has been extended and now supports all three modes: simulation, emulation and live deployment. Just like ProtoPeer, Netbed has a wide range of tools for experimental control and event injection. However, Netbed's simulator uses *ns* [202], which is a packet-level simulator and is less scalable than our message-passing approach.

The focus of ProtoPeer is slightly different than that of the existing frameworks. We designed ProtoPeer primarily with the ease of use and rapid application prototyping in mind. Despite its simplicity, the framework is extensible both up, in the application complexity and down, in the simulation detail. This is how we see the ProtoPeer's development progressing in the future.

A.7. Conclusions

We have presented ProtoPeer, a distributed systems prototyping framework that bridges the gap between simulation and live system deployment. The applications are built on top of a simple time and networking abstraction which allows the user to switch from simulation to live deployment without any changes to the application code. This dramatically speeds up the implement-evaluate-reimplement cycle. All the low-level complexities of managing the network sockets, message serialization and queuing are hidden from the application developer. Applications in ProtoPeer can be modularized into peerlets which are reusable, unit-testable and can be composed together to achieve the desired peer functionality. Finally, ProtoPeer's measurement infrastructure, event injection and scenarios allow for systematic evaluation and performance tuning of the complete system.

Publications: [203, 204]

Index

Wojciech Galuba

w@galuba.info ch. de la Prairie 62 Born: 21 Dec 1978
http://w.galuba.info 1007, Lausanne Nationality: Polish
+1 650 681 7877 Switzerland

Profile

- distributed systems researcher with 2 years of industry experience
- extensive background in distributed data storage, data mining and social computing
- passionate about the ideas at the intersection of social and computer networks
- avid experimenter that enjoys measuring and optimizing systems

Professional Experience

Summer 2010
Research visitor, Social Computing Lab, HP Labs, Palo Alto, CA

Created a highly predictive model of social influence on Twitter. Solely responsible for building a real-time data mining infrastructure processing 20 million tweets a day. Paper published on the topic had over 20 thousand readers and numerous press mentions.

Python, Java, Hadoop, HBase, neo4j, Redis, Gearman, node.js

Summer 2008
Research intern, Docomo Euro-Labs, Munich, Germany

Devised a secure scalable routing protocol for mobile ad-hoc networks. Developed a visualization sandbox for protocol evaluation with MAC layer and radio simulation. Results published at the competitive INFOCOM conference.

2007 - present
Founder and main contributor, protopeer.net

Created a toolkit for rapid distributed systems prototyping. Successfuly deployed on a 800-node testbed. Open-sourced the project and built a community around it. Toolkit used by several universities, 2 EU projects, Docomo Euro-Labs and Nokia.

Java, Python, NIO, Apache MINA, serialization libraries, systems monitoring tools, remote debugging

2002 – 2004
Research engineer, Hottolink Inc., Tokyo, Japan

Lead developer in a half-year 3-person R&D project that combined several open-source libraries into a complete document indexing & recommendation engine. Software subsequently sold to a larger business.
Solely responsible for devising and implementing several social data mining algorithms and a blogosphere visualization tool later deployed as a widget on a 100,000+ user blogging service.

Java, C++, Postgres, Oracle, JSP, libraries for visualization, graph analysis and natural language processing

Summer 2001
Intern, SynaptiCAD Inc., Blacksburg, VA, USA

Developed a compressed memory-mapped on-disk data structure, subsequently marketed as GigaWave, an important feature in company’s high-end products.

C++ on Windows, Linux and Solaris, at syscall level

Education

2005 - present
Ph.D., Distributed Information Systems
Ecole Polytechnique Fédérale de Lausanne, Switzerland
Focus on distributed data storage, peer-to-peer systems security, social computing
Expected graduation: Q1 2011

1998 - 2002
M.Sc., Computer Science
Gdańsk University of Technology, Poland
Focus on distributed systems and data mining
Dissertation: *"A Prototype of a Distributed Web Indexing Service"*

Research

Experience:

- Social computing and social data mining
- Fault-tolerant, scalable and secure routing in peer-to-peer overlays and mobile ad-hoc networks
- Distributed systems prototyping
- Distributed data storage and replication

Selected publications:

"Influence and Passivity in Social Media"
Daniel M. Romero, Wojciech Galuba, Sitaram Asur, Bernardo A. Huberman, arXiv
Press mentions: MIT Tech Review, BBC, CNET, ReadWriteWeb

"Outtweeting the Twitterers - Predicting Information Cascades in Microblogs"
Wojciech Galuba, Dipanjan Chakraborty, Karl Aberer,
Zoran Despotovic, Wolfgang Kellerer, WOSN'10

"Leveraging Social Networks for Increased BitTorrent Robustness"
Wojciech Galuba, Karl Aberer, Zoran Despotovic, Wolfgang Kellerer, CCNC'10, **Best Paper Award**

"Generic Emergent Overlays in Arbitrary Peer Identifier Spaces"
Wojciech Galuba, Karl Aberer, IWSOS'07, **Best Paper Award**

Extracurricular

Set up an infrastructure for harvesting the CPU cycles from idling computers in the university lab. Contributed over 3 years of computation time for clean energy and cancer research at *WorldCommunityGrid.org*

Avid hiker, skier, cyclist, landscape photographer and bicycle repairman.

Languages

Native Polish, Cambridge CPE certified English, basic French, Russian and Japanese