SCALABLE AND ANONYMOUS GROUP COMMUNICATION

Dong Lin

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2015

Supervisor of Dissertation

_____

Boon Thau Loo
Associate Professor of Computer and Information Science

Graduate Group Chairperson

_____

Lyle Ungar
Professor of Computer and Information Science

Dissertation Committee:

Jonathan M. Smith (Chair), Professor of Computer and Information Science

Matt Blaze, Associate Professor of Computer and Information Science

Nadia Heninger, Assistant Professor of Computer and Information Science

Micah Sherr, Assistant Professor of Computer Science, Georgetown University

SCALABLE AND ANONYMOUS GROUP COMMUNICATION WITH MTOR

Dedicated to my family

Acknowledgment

I am deeply grateful to my advisor, collaborators, friends and family for their guidance and help in the last five years. This dissertation would not be possible without their accompany.

I am extremely fortunate to work with my extraordinary advisor Boon Thau Loo. When I was an enthusiastic yet inexperienced fresh graduate student, Boon generously offered me the opportunity to work on his research project and took me as his advisee. I am very grateful that Boon was willing to give me the opportunity to start my Ph.D. despite all the shortcomings I have got. During my work with Boon I have tremendously benefited from his guidance and encouragement, not only in doing research, but also in presentation skill, passion, diligence and discipline, all of which Boon has been a role model for me. What I have learned in the past five years is invaluable to my future life, and I am deeply indebted to Boon for the opportunity and support he has offered.

I am especially grateful to my committee member and collaborator Micah Sherr. Micah, together with Boon, mentored me in my last project on multicast Tor, which is the topic of this dissertation. Micah has guided me with his extensive knowledge in security and anonymity towards the completion of the project. My work on multicast Tor would not be possible without his tremendous help. And I am fortunate to further collaborate with Micah on other anonymity related projects.

I am very grateful to my dissertation committee chair Jonathan M. Smith and committee members Matt Blaze, Nadia Heninger and Micah Sherr. Jonathan, Matt and Micah are principal investigators of SAFEST project, which funded me throughout my Ph.D. study. SAFEST project familiarized me with anonymity technologies and inspired me to design and implement the anonymous group communication system. Nadia kindly accepted the invitation to my dissertation committee, and gave very useful feedback on my dissertation proposal.

During the course of my Ph.D. study, most of my research work was performed in collaboration with my fellow students. I have collaborated with Harjot Gill extensively on Scalanytics project for two years, during which I benefited a lot from Harjot's expertise in system design and software development. I enjoyed the productive collaboration with Lohit Sarna, Xianglong Han, Tanveer Gill, Cam Nguyen, Robert Mead, Kenton Lee and Trisha Kothari on SP4 project. Gang Xiang has offered tremendous help in our work on SAFEST project. I also collaborated with Yifei Yuan on the NetEgg project. I would like to acknowledge my collaborators here, and thank them for their help and accompany.

Also at Penn, I have had the fortune to meet many great friends who made my five years at Penn an enjoyable experience. Zhepeng Yan and I were classmates at high school, and we were roommates during our first two years at Penn. Yang Li, Chen Chen and I share the same advisor. I also thank my other friends Yifei Yuan, Mingchen Zhao, Wenchao Zhou, Fan Yuan, Lu Ren, Yiqing Ren, Yang Wu, Shaohui Wang, Zhuoyao Zhang, and Meng Xu for brightening my Ph.D. life.

I am very fortunate to meet Chen Zhu, my girl friend, who has brightened my life and supported my Ph.D. study in the past three years. Chen was pursing master program at Penn's Graduate School of Education when we first met. I am surprised and excited to see her become interested in Computer Science and successfully got

enrolled in CIT program at Penn. I am looking forward to our prospective life in the future.

This dissertation is funded in part by the DARPA SAFER Warfighter Communication program, NSF CAREER CNS-0845552, CNS-1117052, and CNS-1218066.

Lastly, but most importantly, I am tremendously grateful to my family for their invaluable support throughout my life. My father, Wei Lin, and my mother, Yanbin Chen, have always been very supportive of my education. It is their encouragement and support that allowed me to focus on research without worrying about life. I am so happy to hear their excitement and pride when I told them about my achievement. I hope I can spend more time with them in the future.

ABSTRACT

SCALABLE AND ANONYMOUS GROUP COMMUNICATION

Dong Lin

Boon Thau Loo

Today's Internet is not designed to protect the privacy of its users against network surveillance, and source and destination of any communication is easily exposed to third party observer. Tor, a volunteer-operated anonymity network, offers low-latency practical performance for unicast anonymous communication without central point of trust. However, Tor is known to be slow and it can not support group communication with scalable performance. Despite the extensive public interest in anonymous group communication, there is no system that provides anonymous group communication without central point of trust.

This dissertation presents MTor, a low-latency anonymous group communication system. We construct MTor as an extension to Tor, allowing the construction of multi-source multicast trees on top of the existing Tor infrastructure. MTor does not depend on an external service (e.g., an IRC server or Google Hangouts) to broker the group communication, and avoids central points of failure and trust. MTor's substantial bandwidth savings and graceful scalability enable new classes of anonymous applications that are currently too bandwidth-intensive to be viable through traditional unicast Tor communication—e.g., group file transfer, collaborative editing, streaming video, and real-time audio conferencing.

We detail the design of MTor and then analyze its performance and anonymity. By simulating MTor in Shadow and TorPS using realistic models of the live Tor network's topology and recent consensus records from the live Tor network, we show that

MTor achieves 29% savings in network bandwidth and 73% reduction in transmission time as compared to the baseline approach for anonymous group communication among 20 group members. We also demonstrate that MTor scales gracefully with the number of group participants, and allows dynamic group composition over time. Importantly, as more Tor users switch to group communication, we show that the overall performance and bandwidth utilization for group communication improves. Finally, we discuss the anonymity implications of MTor and measure its resistance to traffic correlation attacks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The capability to communicate anonymously facilitates democracy and the freedom of speech [31, 51]. In authoritarian states, anonymity provides citizens with the freedom of discussion without fear of punishment or retribution, and it allows users to visit websites that would otherwise be blocked by government censorship. Even in democratic countries, the ability to disassociate users' online behavior with their identities serves a critical purpose, since knowledge that unprotected online behavior may be logged (and potentially later revealed) leads naturally to self-censorship—that is, an unwillingness to access information for fear of future exposure.

Communication participants may wish to hide the fact that they are communicating for a variety of reasons. The uses of anonymity include, but are not limited to, the following scenarios:

- **Whistle-blower protection.** The whistle-blower may want the ability to publish secret information, news leaks, and classified media anonymously without fear of retribution. This includes the publishers of WikiLeaks. The famous whistle-blower, Edward Snowden, also specifically recommended Tor to cover user tracks.

- **Circumvention from firewall blocking.** Tor can be used to allow constituents of repressive regimes to freely access websites and services that are otherwise blocked by the firewall. For example, users in China and Iran use Tor to access websites such as Facebook, Twitter, Google, etc.

- **Privacy protection.** Anonymity protects users from being associated with any sensitive activity that may otherwise implicate or embarrass the users. These sensitive activities include search on Google, access to health information, discussion in online group, etc.

However, today's Internet is not designed to protect the privacy of its users against network surveillance. Even though robust cryptographic techniques are available to prevent unauthorized parties from learning the communication contents, the fact that two parties are communicating is easily discernible. The IP packets carry the source and destination addresses of any communication in plain text which are used by intermediate routers to route the packet. Therefore, the participants as well as time and duration of the communication are easily exposed to eavesdroppers.

Various techniques have been proposed to meet the demand for anonymous communication on the Internet [15, 45, 7, 41, 9]. Such approaches typically route messages through intermediate relays before delivering them to the intended destination, such that no message will carry the actual source and destination addresses at the same time. These techniques usually vary in their anonymity guarantee, performance, and application domain.

1.1 Limitations of Existing Anonymity Systems

There is a strong need for anonymity systems that provide users with practical performance and ease-of-use while avoiding central point of trust. A large body of anonymity systems have been proposed with varying anonymity and performance properties. On one end of the spectrum, there are systems that offer provably strong anonymity guarantees [9, 10] even against well-positioned and well-provisioned adversaries. But these systems incur high communication and computation costs, and are best applied when the participants in a communication are fixed and are small in number. On the other end of the spectrum, low-latency anonymity systems such as Tor [15] offer the strongest practical anonymity currently available. But Tor is currently constrained to provide anonymity only for unicast communication.

The increasing demand for practical anonymous group communication has spurred a large crowd of commercial ventures. For example, popular anonymous applications are available on iOS and Android devices that allow users to gossip anonymously with their friends [44], with nearby users [60], or with all users of the system [59]. Facebook recently unveiled its own anonymous application [43] to foster anonymous group discussion among people with similar interests. However, these solutions introduce a single point of trust, since one compromised server—or one subpoena—can break users' anonymity. This threat is not merely academic: Whisper [59] was recently reported to silently track and monitor their users' locations [52].

Anonymous group communication can be straightforwardly achieved using traditional unicast anonymity networks (e.g., Tor) and an external facilitator. Here, group members anonymously send their messages to the facilitator, which then "echoes" the messages to other group members. However, such an approach incurs unnecessary bandwidth and latency overheads, and scales poorly with group size. More

importantly, approaches still render users' anonymity vulnerable to the operator of the external facilitator.

This dissertation presents MTor, a practical anonymous group communication system that supports dynamic group composition with scalable performance. We construct MTor as an extension of Tor, benefiting Tor's large user base by allowing them to perform anonymous group communication and enabling new types of anonymous applications (e.g., multi-party audio conferencing), while also benefiting from Tor's network infrastructure as well as its mature design and implementation. MTor constructs multicast trees of Tor relays across group participants. Any user can enter and leave the group communication without global coordination by joining as leaves to these trees. To the best of our knowledge, MTor is the first system providing low-latency anonymous group communication with a decentralized trust infrastructure.

1.2 Research Questions

In comparison to existing non-anonymous multicast protocol, the design goal of anonymity in a potentially adversarial environment imposes unique requirement on the protocol. Specifically, the multicast protocol should 1) construct multicast tree randomly in a decentralized manner to reduce risk of targeted attack; 2) integrate with state-of-art technique in authentication, cryptography, anonymity, etc.; and 3) achieve acceptable anonymity under proposed threat model. To meet these requirements, this dissertation explores the following research questions:

- **How to build a randomized multicast tree across clients without global coordination?** Existing multicast protocols typically construct multicast tree by having clients issue join request to a static node, which acts as rendezvous point

for the group. However, such approach is not suitable for anonymous communication in an adversarial environment, since it allows adversary to easily locate and take over rendezvous point and other nodes on the tree to de-anonymize clients. This dissertation provides a mechanism for clients to construct a randomized multicast tree in a semi-decentralized manner.

- **Can we integrate anonymous group communication system with some existing overlay network to leverage its secure implementation and existing developer community?** Instead of building MTor from scratch, we choose to build it as an extension of Tor, to benefit from Tor's mature design, developer community and large number of volunteer relays. This also allows us to benefit Tor's large user base by scaling up performance of existing multi-party communication and enabling new types of anonymous communication (e.g. multi-party audio conferencing). We detail the design and implementation of MTor in the rest of the dissertation.

- **What are the performance and anonymity of MTor if it is deployed on the live Tor overlay network?** In addition to measuring bandwidth savings typically evaluated for non-anonymous multicast protocol, we also evaluate the latency improvement experienced by end users, probability of communication disruption due to relay failure, as well as compromise rate against traffic correlation attack. To accurately estimate MTor's performance on live Tor network, our evaluation takes advantage of state-of-art Tor's evaluation methodologies combined with realistic models of Tor overlay network, user behavior, historical datasets of Tor relay information, and a prototype implementation of MTor in C++.

1.3 Contributions

In the rest of the dissertation we address the research questions listed in Section 1.2. This dissertation makes the following contributions:

- The semi-decentralized design of a practical communication system that enables anonymous group communication without central point of trust in a potentially adversarial environment. The system provides best-effort reliability guarantee and automatically adapts to changes in group membership and underlying network.

- A multicast protocol for Tor that saves bandwidth and scales up performance for both one-to-many and many-to-many anonymous communication.

- Definition and evaluation of compromise due to traffic analysis attack in the group communication setting.

- A prototype implementation and detailed evaluation of MTor using techniques from recent research and latest consensus records from live Tor network. We compare performance of MTor with baseline approach which combines vanilla Tor with an external service, and demonstrate how MTor enables an important class of communication service – anonymous group communication with low-latency delivery requirement.

1.4 Organization

The rest of the dissertation is organized as follows. Chapter 2 discusses research in related areas. Chapter 3 gives an overview of MTor's motivation and challenges.

Chapter 4 presents MTor's threat model. Chapter 5 describes the design and implementation of MTor. We present performance and anonymity evaluation results in Chapter 6 and conclude the dissertation in Chapter 7.

Chapter 2

Related Work

The techniques presented in this dissertation expand upon prior work from the security, anonymity, and networking communities. This chapter introduces the terminologies used in this dissertation, and overviews related research in these areas.

2.1 Terminology

We adopt the following definitions throughout this dissertation. In addition to traditional definition of traffic correlation attack in unicast context, we define two subclasses of traffic correlation attack in the group communication context. We define anonymity metrics related to these definitions of traffic correlation attack in Section 6.2.2.

Definition 1. *(Traffic Correlation Attack) Traffic correlation attack attempts to associate the sender with the receiver of the communication by correlating the footprint (e.g. time and volume) of observed traffic entering and exiting the anonymity network.*

Definition 2. *(Linkage Attack) In the linkage attack, an adversary attempts to*

determine if two given clients are communicating with each other. Depending on the design of anonymous communication system, the adversary may or may not need to observe traffic of both clients simultaneously.

Definition 3. *(Membership Identification Attack) In the membership identification attack, an adversary attempts to find out if a given client belongs to a given group. A necessary condition is for adversary to observe traffic entering or leaving the given client.*

Tor and other low-latency anonymity systems are known to be vulnerable to traffic correlation attack. In this dissertation, we focus on improving Tor's performance while still providing anonymity against linkage and membership identification attack.

2.2 Tor Anonymity Network

In the following, we provide details on the mechanics of Tor that are relevant to our proposed techniques.

Tor operates as an distributed overlay network consisting of approximately 6200 volunteer-operated relays. It is designed to provide low-latency anonymous connection to TCP-based applications like web browsing, instant messaging, and ssh connection. It uses source-based routing that tunnels messages through a number of intermediate routers before delivering messages to the destination. Messages are multiply encrypted, such that intermediate routers can only identify the previous and next hops in the path. Tor is designed based on onion routing [49], with its own improvement on security and performance.

Before transmitting data to destination, the source Tor client constructs anonymous path (i.e. circuit) as a sequence of Tor routers. For each router on the path,

the client sends a CREATE cell that carries a random seed, and a mutually shared symmetric key is generated based on the seed. These symmetric keys are later used for encrypting DATA cells.

The source Tor client begins data transmission once it finishes constructing the anonymous path. The data from user application is repeatedly encrypted using the symmetric keys previously negotiated with each router on the path, applied in reverse order. For example, suppose keys k_1, k_2, k_3 denote the symmetric keys of the first, second, and third routers, then data M is encrypted as $\{\{\{M\}_{k_3}\}_{k_2}\}_{k_1}$ into DATA cell. Upon receiving a DATA cell, each router decrypts the outermost layer, extracts IP address of the next hop in the outermost layer, and forwards the cell to the next hop. The last router on the path delivers unencrypted data to destination.

The downstream data is encrypted in the reverse order. Upon receiving a DATA cell, each router encrypts the cell using its symmetric key, looks up the anonymous path associated with the DATA cell, and forwards data to the next hop in the downstream direction. After receiving the DATA cell, the source Tor client repeatedly decrypts the data before forwarding cell payload to user application.

Tor is known to be slow in terms of performance [16] (though there is evidence that its performance is improving [53]). The slowness is not attributed to fundamental design flaws in Tor, but rather, mostly attributed to a large asymmetry between the number of clients who wish to use the network and the relays who route their traffic. There are a large number of proposals to improve the performance of Tor, including techniques for improving path selection [46, 57, 1, 48], providing incentives for users to run relays [33, 36, 28], reducing congestion [26, 2], and modifying Tor's transport protocol [40] and circuit scheduling algorithm [50]. MTor may also benefit from many of the above techniques, but additionally explores methods of de-duplicating information by leveraging traffic aggregation in multicast trees. To

the best of our knowledge, MTor is the first to introduce a multicast primitive for Tor.

2.3 IP multicast and Overlay Multicast

IP multicast [11] provides efficient group communication at the network layer by reducing message duplication on physical links. However, it requires router support at the infrastructure level and adds complexity by requiring routers to maintain per group state. Multicast is also susceptible to denial of service attacks, since by design, it amplifies messages by number of receivers. As a result, IP multicast is not widely deployed.

Unlike IP multicast, overlay multicast provides multicast service at the application layer. As overlay multicast does not rely on router support, it allows multicast functionality to be incrementally deployed on the existing network. The key concern regarding overlay multicast is the performance penalty involved in disseminating data using overlays rather than native IP multicast. A number of systems have been proposed to provide efficient overlay multicast, including Scribe [5], Narada [8], Overcast [24], and Yoid [17].

However, existing overlay multicast techniques do not provide anonymity for low-latency group communication. Overcast [24] builds a single-source multicast tree where the operator of the root knows identity of all group members. Narada [8] forms the multicast tree by building a mesh per group containing group members before constructing a spanning tree of mesh for each source. However, its mesh creation and maintenance algorithms assume that all group members know about each other. Scribe [5] is the most similar to MTor. It builds multicast tree in a fully decentralized manner by using Pastry, a peer-to-peer location and routing substrate,

to route packets. Multicast tree is formed by joining Pastry routes from each group member to a rendezvous point associated with the group. However, the rendezvous point is static after it is chosen, and the path from client to rendezvous point is fully exposed to any observer. In comparison to these efforts, MTor provides anonymity-aware multicast service over an existing anonymous overlay network (Tor).

2.4 Anonymous Multicast

A number of existing anonymous multicast schemes provide provable (unconditional) anonymity guarantees, but at the cost of limited performance or requiring that the group's composition be static. Classic DC-nets [6, 10] provides provable anonymity even against traffic analysis attack. But communication and computation costs have in practice limited its performance and anonymity set size. Herbivore [20] supports mass participation by securely dividing large networks into smaller DC-nets groups, but guarantees anonymity only within each group, showing only scalability to 40-node groups. Dissent [9] significantly improves the scalability and performance of DC-nets by switching to a client/server architecture. But Dissent's protocol halts completely even if a single server goes offline; and its group composition cannot change after the initial setup. Furthermore, Dissent's performance relies heavily on a small set of physically co-located servers with high-bandwidth and low-latency communication among them, which reduces performance for geographically distributed clients. In contrast, MTor provides wide-area anonymous group communication with dynamic group composition and achieves better performance than that offered by Tor.

There is also related work that examines multicast for low-latency anonymity

networks. M2 [38] offers receiver anonymity for one-to-many multicast communication. However, M2 assumes mutually-trusting receivers, and does not protect the identity of the sender. Therefore it is not useful for many-to-many anonymous group communication.

Hordes [47] is a variant of Crowds [41] that leverages IP multicast for return traffic from the receiver to the sender. Unlike MTor, Hordes uses multicast as an anonymous transport mechanism for sending unicast messages, in a way that hides the identity of the intended recipient. However, this approach wastes bandwidth by delivering the majority of messages unnecessarily to unintended recipients. In contrast, MTor saves bandwidth and provides scalability for anonymous group communication.

2.5 Relay Selection in Tor

Tor offers anonymity by routing messages through multi-hop path on an application-layer overlay network. But doing so typically yields significantly worse performance as compared to standard IP routing. Each intermediate relay on the anonymity path potentially increases the latency and limits the bandwidth of the end-to-end communication. The expected performance of Tor therefore depends on the metrics of selected relays on the path: the throughput is determined by the relay with least bandwidth, the latency is the sum of latencies between adjacent relays, etc.

The relay selection strategy affects both performance and anonymity of the communication. To maximize anonymity, Tor client can choose relay uniformly at random, leaving no information to adversary on which to bias probability distribution over the candidate relays. However, such an approach tends to result in path with poor performance.

On the other hand, Tor client can choose to optimize relay selection for performance by preferentially selecting those relays with high throughput and stable connectivity. Such performance-aware strategy, however, imposes non-uniform distribution over candidate relays. In comparison to uniform relay selection, it leaves opportunity for adversary to discover selected relays with higher probability. Furthermore, an resourceful adversary can run relay with high bandwidth as well as low latency to monitored target, so that his relay is more likely to be selected on the anonymity path.

Therefore, the relay selection strategy involves trade-off between performance and anonymity. Unlike other anonymity systems whose relay selection is controlled by the overlay network (e.g. Crowd [41]), in Tor the sender has full control over relay selection, which allows it to adjust position in the anonymity-vs-performance spectrum [48, 45] to meet the requirement of different applications.

We briefly describe the default relay selection strategy in Tor. The Tor client first fetches a list of candidate relays (i.e. consensus document) from one of the Tor directory server, in which each relay may be marked with flags STABLE (having a sufficiently long uptime), VALID (running a recent version of Tor), FAST (having sufficient bandwidth), GUARD (can be the first node on the path), and EXIT (can be the last node on the path). Those relays with EXIT flag further provide exit policy to specify the range of IP address and port that it allows to connect to. To achieve reasonable trade-off between performance and anonymity, Tor uses a bandwidth-weighted relay selection strategy, where relay is selected with probability proportional to its bandwidth. The sender repeats this strategy to select each relay on the path.

Chapter 3

Overview

MTor provides a group communication primitive in which any member of a multicast group may originate messages; these messages are tunneled through anonymous Tor circuits to all other group members. Our security goal, which we describe in more detail below, is to prevent an adversary from (1) discerning the sender of the message and (2) enumerating the members of the group.

We envision that any number of such groups (including zero) may exist on Tor at any given time. To participate in a group, we assume that group members obtain a succinct *group descriptor* document that contains the group's name, its unique group identifier (GID), and an optional certificate. (Additional fields in the group descriptor are described in Section 5.1.) The certificate enables an access control mechanism for multicast and is useful to enable single-source multicast streams in which messages are only relayed if they originate from the group's authorized source. (Although we do not consider it here, MTor can be trivially extended to support ring signatures [42] or other cryptographic structures that permit multiple authorized senders for a group.) Omitting the public key enables a multicast group in which any group member can act as both sender and receiver. Finally, the group identifier is

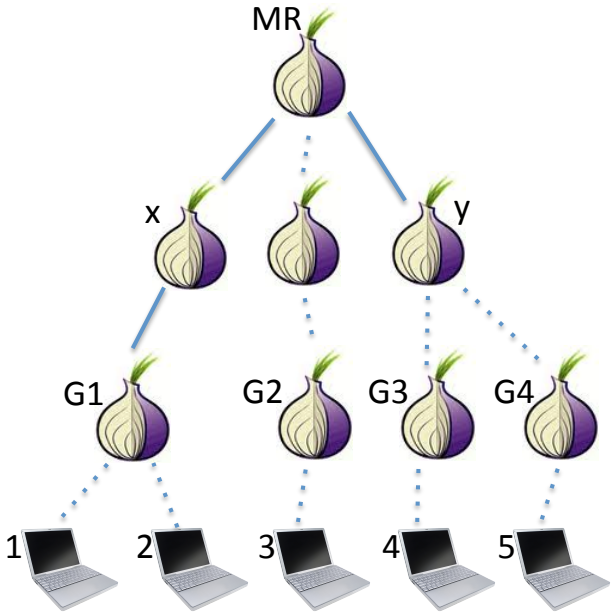


Figure 3.1: Example multicast tree.

computed as a hash over the group descriptor's attributes in a manner that binds the (optional) certificate to the group. We assume that the group descriptor is retrieved anonymously by the group's participants—i.e., it may be communicated out-of-band or it is retrieved over a standard unicast Tor circuit.

Every hour, each member of a given group individually runs a local deterministic algorithm that takes as input the group's GID and the local Tor client's cached copy of the Tor consensus document (the list of available relays), and outputs a Tor relay that will serve as the root of the group's multicast tree. We call that root the *multicast root* or MR. Importantly, all group members compute the same MR as long as they have the same up-to-date Tor consensus document from one of the Tor directory authorities.

A significant advantage of MTor is that multicast allows for aggregation of messages, which eliminates redundancy and conserves bandwidth. To illustrate, Figure 3.1 shows an example multicast tree constructed with MTor. Here, five clients

(1-5) construct three-hop Tor circuits to the MR. As with unicast Tor circuits, clients begin their circuits with fixed guard relays (G1-G4). In contrast to normal Tor communication, MTor aggregates identical traffic that flows across a single connection between relays. For example, consider the case in which client 3 sends a message to the group. The message is sent via a Tor circuit to the MR, which then forwards it to its two neighbors (relays "x" and "y"). Only one copy of the message is sent for each connection. This preserves bandwidth, since x and y are each servicing two downstream clients. Similarly, bandwidth is again preserved when x forwards along the message to G1, despite the presence of clients 1 and 2 on the multicast tree.

Notably, MTor does not rely on *exit relays*—relays that serve as egress points in the Tor network. In MTor, *all* traffic is sent within Tor. Exit relays are unnecessary in MTor since traffic never leaves the Tor network. In a mixed Tor network that carries both unicast and multicast traffic, this is a desirable feature: exit relays (which are especially valuable since they constitute only approximately 1/3 of all relays in the live Tor network) can be reserved for unicast traffic.

Summary of results We evaluate MTor using Shadow [27], a high-fidelity discrete-event simulator that runs actual Tor code on a synthetic network topology. Shadow has recently been used to evaluate Tor's circuit scheduling algorithms [27, 26], its vulnerability to traffic correlation attacks [30], and proposed performance enhancements [29, 18]. By simulating MTor's path selection algorithm using historical records of Tor's consensus documents, we evaluate the bandwidth consumption and anonymity of MTor. Our results are encouraging: for large sized groups, MTor achieves 62% savings in network bandwidth as compared to vanilla Tor; even for smaller-sized group of 20 clients, MTor achieves 29% savings in network bandwidth

and provides significantly improved client experience, decreasing the median transmission time of message delivery by as much as 73%.

Chapter 4

Threat Model and Assumptions

We adopt Tor's threat model in which an adversary monitors or controls some fraction of the network [15]. For example, the adversary may operate Tor relays, or may monitor or control some portion of the underlying Internet. We assume the adversary cannot monitor all communication, since Tor is not designed to protect against global adversaries [15]. Finally, we conservatively assume that the adversary has access to the group descriptor document and can effectively join any multicast group.

Tor is known to be vulnerable to an adversary who can observe and correlate traffic entering and exiting the anonymity network. This type of *traffic correlation attack* is arguably the most serious known de-anonymization attack against Tor [21, 55] and recent studies have demonstrated that even a moderately provisioned adversary can de-anonymize most Tor users within a few months [30]. In this paper, we focus on such traffic correlation attacks since, when successful, they directly identify an anonymous communication's endpoints and defeat Tor's anonymity goals (i.e., to conceal communicants' network locations).

We do not consider MTor's resilience to attacks that enumerate the relays involved in anonymous communication, since merely discovering which Tor relays were involved in an anonymous communication does not reveal the participants of that communication. We emphasize that such "path discovery" attacks are trivially achievable in vanilla Tor by an adversary who operates a Tor relay and is chosen as the middle relay in an anonymous circuit; here, the malicious relay immediately learns its neighbors (i.e., the guard and the exit) and thus discovers the entire anonymous path. Importantly, learning the relays involved in an anonymous path does *not* by itself identify the network locations of the Tor client or the destination and thus does not break anonymity. This is in contrast to traffic correlation attacks—the focus of our security analysis—which *do* reveal the communicating parties.

Since, in MTor, messages may have multiple recipients, we consider two variants of a traffic correlation attack: We consider an adversary's ability to determine whether a given client is participating in a multicast group. If the adversary can monitor that client's (encrypted) communication with its guard, then we assume that the adversary can apply simple traffic analysis techniques to determine that the client is a subscriber of the group. Second, we consider attacks in which the adversary is able to identify both the receiver and sender of a multicast message; here, the adversary must monitor both clients' communications to correlate traffic. We evaluate how MTor affects an adversary's ability to perform traffic correlation attacks in Section 6.2.

Finally, we assume a computationally-bounded adversary who cannot find collisions or preimages of cryptographic hashes, decrypt messages without knowledge of the decryption key, or forge digital signatures. Since MTor uses Tor as its backbone, we additionally assume that Tor's existing transport protocol is secure (e.g., that keys are randomly generated, that ciphers are strong and used correctly, that

the implementation is correct, etc.). MTor does not impose any restrictions on the "last mile" connection between the client and the first relay (i.e., a guard or bridge) and is compatible with Tor's pluggable transports and obfuscated bridges (see, for example, [58]). We therefore consider local eavesdropping attacks such as website fingerprinting [22, 37, 4] orthogonal to this work, since solutions [4] and mitigations [54] intended for vanilla Tor are also applicable to MTor.

Chapter 5

Design

To support group communication, MTor constructs a multicast tree at the application-layer using Tor relays as the internal nodes of the tree. The leaves of the tree are clients, who connect through their guard relays (i.e., the clients' guards are the parents of the clients in the tree). We emphasize that MTor does not use IP multicast, and instead uses Tor's existing SSL/TLS transport mechanism between relays to provide link-level authentication and confidentiality. The multicast tree is constructed in a dynamic and decentralized fashion, and does not require global coordination.

In this chapter, we describe in detail how MTor constructs and maintains multicast trees to ensure correct functionality and provide efficient group communication.

5.1 Group Descriptors

Before a client can participate in a group communication, it needs to obtain a *group descriptor* for that group. We envision that the group descriptor could be communicated through some out-of-band mechanism—for example, via emails or a distributed key-value store—and can be retrieved anonymously (e.g., by using Tor).

The group descriptor contains the following attributes:

- **Group name.** The group name is a human-readable string (e.g., "Freedom Radio") that is intended to describe the group's purpose. We do not require that the group name be unique since the group identifier (described below) is calculated based on the entire descriptor; however, unique group names provide easier distinguishability between advertised groups.

- **Bandwidth.** The bandwidth attribute specifies a minimum bandwidth that relays must support to be a member of the multicast tree. A conservative bandwidth estimate prevents message loss, which is possible in MTor when there are bottlenecks in the multicast tree. Message loss is discussed in greater detail in Section 5.6.

- **Certificate.** When present, the optional certificate attribute contains an X.509-encoded certificate containing a public key and validity date. The public key could be self-signed—in which case group messages are authenticated according to a trust-on-first-use policy—or the key could be signed by an external party, allowing the use of a PKI. If a certificate is present in the group descriptor, relays that receive group messages will verify that those messages have been properly signed before they are forwarded. We describe the authentication mechanism, as well as its overheads, in more detail in Sections 5.4 and 6.1.8.

- **Cipher identifiers, confidentiality key, and MAC key.** In contrast to unicast Tor, MTor does *not* by default offer any end-to-end confidentiality guarantees. This is necessary to allow bandwidth savings via link aggregation and message de-duplication. However, it also implies that any relay who is part of the

group's multicast tree can eavesdrop on the communication. These optional cipher attributes provide a simple mechanism for secure end-to-end communication. However, *confidentiality relies on the secure dissemination of the group descriptor file*. Users who have access to the descriptor can protect the confidentiality of their messages by encrypting them with the symmetric confidentiality key and appending a MAC. Eavesdroppers who do not have access to the descriptor cannot learn the plaintext of the group messages. These confidentiality extensions are used only at the endpoints (to encrypt and decrypt messages) and therefore do not incur any additional computational cost at Tor relays.

The group descriptor file is hashed to produce a *group identifier* (GID) that uniquely and concisely identifies the group. Specifically, the GID is calculated as

$$\text{GID} = h(\text{group name}|\text{bandwidth}|\text{certificate}|$$
$$h(\text{cipher identifier and keys}))$$

where h is a cryptographic hash function and | denotes concatenation. For each Tor cell being sent via multicast to a group, we include the group identifier that uniquely identifies its corresponding group. The group identifier is also used to select a multicast root (MR) for the group, which is described next.

GID binding proofs The construction of the GID allows for a *GID binding proof*, where a prover provides the GID, the group name, the bandwidth, the certificate (if present), and a hash over the cipher identifier and keys (if present). Importantly, the GID binding proof does *not* reveal any keys. The verifier then computes the GID from the provided inputs and verifies that the computed GID matches the provided GID. (We operate in the random oracle model and assume an ideal hash function, which we approximate in our implementation using SHA hashes.) GID

binding proofs are used to bind a certificate to a GID, and enable authenticated group communication, as described in Section 5.4.

5.2 Multicast Root Selection

Algorithm. SelectMR(min_bw, GID**):**
1. $cur_hour \leftarrow$get_current_hour()
2. fetch the *cons* from cache/directory server such that:
   $cons \leftarrow \underset{cons}{\text{argmin}}\{\text{get\_valid\_after}(cons) \mid cons \in \text{get\_recent\_cons}() \text{ and } \text{get\_valid\_after}(cons) \geq cur\_hour\}$
3. for $relay \in cons$: /* **construct ring** */
3.1 if is_stable($relay$) and is_fast($relay$) and get_bandwidth($relay$) $\geq min\_bw$:
3.1.1 $relay_{\text{pos}} \leftarrow \text{hash}(relay_{\text{digest}} + GID + cons) \mod 2^{160}$
3.1.2 put $relay$ on the $ring$ at $relay_{\text{pos}}$
4. $begin_{\text{pos}} \leftarrow \text{hash}(GID) \mod 2^{160}$
5. $relay \leftarrow \text{find\_next}(begin_{\text{pos}}, ring)$
6. while true: /* **search for the first active relay** */
6.1 if create_circuit_to_mr($relay$) == success:
6.1.1 return $relay$
6.2 $relay \leftarrow \text{find\_next}(relay, ring)$

Figure 5.1: MR selection algorithm. find_next($X, ring$) returns the relay on the ring whose identifier is the least greatest than X, modulo 2^{160}.

To enable group communication, MTor forms multicast trees over the Tor relays. Clients join a group by forming circuits to the root of the desired group's multicast tree—i.e., the *multicast root* (MR). Thus, MTor requires a mechanism for ensuring that clients who wish to join the same group select the identical MR. More concretely, the MR selection algorithm should meet the following criteria:

- *Correctness:* all clients in a given group must agree on the same MR regardless of their startup time and location, to ensure the multicast tree spans across all clients during group communication.

- *Anonymity:* clients should select the MR without relying on global coordination, or more generally, without disclosing their network location.

- *Efficiency:* MR selection should (i) introduce little or no overhead to the Tor network, (ii) be stable enough for persistent group communication, and (iii) choose a relay that has sufficient bandwidth to not be the bottleneck for group communication.

One straightforward solution is for the group initiator to register MR information in a lookup service that all other Tor clients can access, in much the same way that Tor hidden services register and advertise their introduction point [56]. This solution is easy to deploy and does not introduce any overhead to the Tor network. However, this requires exactly one client to be designated to monitor and update the MR throughout the group communication. We desire a more flexible approach that allows for MR-migration (that is, switching the MR from one relay to another) and does not require the client that originated the group messaging session to stay online for the session's duration.

MTor uses an alternative design that leverages Tor's existing infrastructure. In Tor, clients periodically retrieve a *consensus document* that lists the available relays, their public keys, network addresses, exit policy, status, and other information. These documents are polled either from authoritative directories—which undergo a voting protocol to form the (digitally signed) consensus—or directory caches. In either case, clients authenticate the consensus document by verifying that it has been signed by a majority of the directory authorities.[1] As its name implies, the consensus document should be approximately consistent among all clients. To mitigate edge cases (e.g., in which a client retrieves the consensus moments before the directory

[1]Digests of the directory authorities' public keys are hard-coded with the Tor client.

authorities generate a new consensus), MTor selects MR from the oldest available consensus whose `valid-after` attribute is larger than the current hour time. We note that Tor directory authorities can support such consensus requests with only minor modification.

In MTor, clients independently select the MR using a local variant of consistent hashing. Since the MR is a central point of failure in the multicast tree, MTor first applies a filtering process to weed out undesirable relays. Only relays that have earned the STABLE and FAST flags (respectively indicators of stability and performance) and can provide at least the bandwidth specified in the *group descriptor* are considered. The remaining Tor relays are then logically organized in a ring over $[0, 2^{160})$, with each relay's value in the ring being equal to a hash over its digest (a fingerprint of the relay's public key), GID[2], and consensus document used for the MR selection.[3] Note that these "rings" are computed locally for each client using only local knowledge and a cached copy of the consensus. The client selects the MR by finding the relay whose value in the ring is the least greater (modulo 2^{160}) than the GID. The client then attempts to create a unicast Tor circuit to the MR (the mechanism for selecting relays in this circuit is described in Section 5.3). If it is unsuccessful, then the next closest value in the ring is considered the MR, and this process repeats until a live candidate relay is discovered. The complete MR selection algorithm is more formally presented in Figure 5.1.

[2]The GID is included to evenly distribute MR of different groups across relays.

[3]The consensus document is included in the preimage to prevent malicious relays from generating public keys that yield generally advantageous positions in the ring. Since each communication session has an unpredictable consensus, this effectively "randomizes" the placement of relays in the ring for each group. Note that malicious relays cannot easily regenerate keys to find an advantageous position for a current or incoming communication session, since the act of regenerating its keys will cause it to lose the STABLE flag and consequently become excluded from consideration.

5.3 Tree Formation

A client joins a multicast group by constructing a Tor circuit to the group's MR. The procedure for building such a circuit is similar to normal circuit construction in Tor, with the exceptions that (1) the MR is used in place of an exit relay; (2) to prevent certain deadlock conditions, we restrict the set of potential middle relays; (3) each relay on the circuit must provide at least the bandwidth specified in the *group descriptor*; and (4) if the any relay is already in the multicast tree (e.g., selected by other group members), the client stops circuit construction and uses whatever is upstream from that relay.

Relay selection A client who wishes to use group communication first either establishes a new group by creating a new group descriptor or obtains an existing group descriptor through some out-of-band mechanism. Next, the client selects a 3-hop circuit, consisting of one of its guard relays, followed by a middle relay, and ending with the MR. All three relays should provide at least the bandwidth specified in *group descriptor*. The middle relay is selected using Tor's default bandwidth weighting strategy, except that the client enforces that the middle relay has a higher digest than that of the guard relay. This latter constraint prevents our distributed tree construction algorithm from running into deadlock.[4] Note that exit relays are not necessary here since all group communication is carried over Tor's SSL/TLS connections, and never "exits" the anonymity network. We remark that the clients who opt to use MTor for group communication rather than constructing multiple unicast circuits (each of which consumes bandwidth at exit relays) are effectively saving valuable exit relay bandwidth for non-group communication.

[4] As an example, consider the case where client A selects path $x \to y \to MR$ and client B selects path $y \to x \to MR$. The algorithm might deadlock if two the clients begin path construction at roughly the same time.

Tree construction After the guard, middle, and MR relays have been selected, the client starts constructing the circuit to the MR by sending a `CREATE` cell with the GID and a GID binding proof to its chosen guard. (In Tor, `CREATE` cells signal the creation or extension of an anonymous circuit.) The circuit is similarly extended to the middle and then the MR by tunneling additional `CREATE` cells, again including the group identifier and a GID binding proof. However, if any relay is already in the multicast tree (e.g., selected by other group members), the client stops circuit construction and uses whatever is upstream from that relay. In effect, clients' MTor circuits may contain fewer than three hops if either the chosen guard or the middle relay is already forwarding messages for that multicast group.

To support forwarding of multicast messages, each relay maintains a local key-value store called the *multicast forwarding table* that is keyed on the group identifier (which is communicated through `CREATE` cells) which contains routing information for a group.

Upon receiving a `CREATE` cell, a relay looks up the included group identifier in its multicast forwarding table, and responds as follows:

- If the relay has not previously received a `CREATE` cell, it replies with a `CREATED` cell, mirroring Tor's default behavior. After receiving the `CREATED` cell, the client will continue its path construction towards MR via this relay.

- If the relay has already received a `CREATE` cell from another client, it replies with a `HOLD` cell, indicating that tree construction is already under progress. After receiving the `HOLD` cell, the client will wait for a `BEGIN` cell. The source of the `BEGIN` cell is described below; conceptually, it signals that the tree has been created.

- If the relay has already received a `BEGIN` cell, it replies with a `BEGIN` cell, indicating that the tree construction is completed. The client can now start group communication.

In all cases, the relay records in its multicast forwarding table the *previous hop* from which it received the `CREATE` cell and the *next hop* to which it forwards the `CREATE` cell. This information represents the relay's parent and children in the multicast tree. The table is later used to forward messages during group communication.

Lastly, exactly one MTor client will receive a `CREATED` cell from MR. When that happens, the client informs MR of its role, which then multicasts a `BEGIN` cell across the multicast tree, to inform all relays and clients on the multicast tree to start group communication.

5.4 Sending and Receiving

A client can begin sending and receiving group messages once it has received the `BEGIN` cell. Outgoing messages are sent via the client's Tor circuit towards the MR. In MTor, messages should traverse each edge in a multicast tree only once. When a relay receives a message, it looks up its neighbors in the tree by searching its multicast forwarding table for the records that are keyed by the group identifier. The incoming message is then forwarded to the relay's adjacent edges, excluding the message's incoming edge. Group messages percolate down the multicast tree, and are eventually delivered by guard relays to the subscribed clients.

MTor has the potential to offer significant bandwidth savings for group communication as compared to unicast-based approach. Consider, for example, a strawman solution based on vanilla Tor in which clients use an external service such as a bulletin board, IRC server, or Google Hangouts to aid in group communication. The

service supports group communication by "echoing" incoming messages to all connected clients (i.e., through their Tor connections). MTor uses significantly less bandwidth than this unicast-based approach, since the former (i) offers the possibility of message de-duplication by aggregating data on shared links in the multicast tree, (ii) uses a single multicast root rather than multiple exit relays, which both frees up exit relay resources and reduces the number of relays that are involved in the group communication, and (iii) avoids the overhead of communicating with the external service. In Section 6.1, we empirically measure these bandwidth savings under realistic network conditions and workloads.

Message confidentiality If the group descriptor contains a cipher identifier and encryption key, then all group messages are presumed to be encrypted by the messages' senders. Receivers use the decryption and MAC keys from the group descriptor to respectively decrypt and authenticate messages. Our design is purposefully flexible and allows the creator of the group to specify the symmetric key cipher and MAC algorithm. Importantly, this "end-to-end" encryption of group messages is transparent to Tor relays, since messages are encrypted/decrypted only by the group members.

Authentication and DoS prevention When the group descriptor includes a certificate, MTor provides a weak form of authenticated multicast: only clients that have knowledge of the private key that corresponds to the certificate may send messages. Clients sign their cells, storing the signature and a timestamp (to prevent replay) as added fields.

Recall that relays are given both the GID and a GID binding proof, and hence relays can extract the certificate (if it exists) from the proof. Relays enforce authentication by verifying received cells' signatures and dropping cells that fail verification. This mitigates potential DoS attack against the Tor network by preventing both malicious clients and relays from propagating unauthentic messages.

A clear disadvantage of using the above authentication scheme is that it incurs significant bandwidth and computational overheads. We propose two potential solutions to reduce these overheads: First, we can reduce bandwidth overheads by relying on short signature schemes such as ECDSA which offers equivalent security to a 2048-bit RSA signature using only a 283-bit public key [3]. Second, for single-source streaming multicast messages, the sender may transmit special signature cells that contain a signed list of upcoming (yet-to-be-received) cell hashes. After receiving a signature cell, a relay verifies the signature over the hashes, and then verifies that the cells it subsequently receives have those hash values. Since the cost of verifying a forged packet is relatively low and an adversary has a very limited opportunity of finding a collision, MTor can use truncated hash digests. For example, if hashes are truncated to 40-bits, then a single 512-byte signature cell can hold 91 40-bit hashes, a 283-bit ECDSA signature, plus 18-byte header, reducing the verification and storage cost by nearly two orders of magnitude.

5.5 Churn Handling

To effectively detect and recover from link or relay failures, MTor maintains multicast tree states (i.e., its upstream and downstream links) as soft-state in each relay. The MR periodically multicasts heartbeat cell across the tree. The relay receiving the heartbeat cell will refresh the table entry for the incoming link; and the relay successfully sending the heartbeat message will refresh the table entry corresponding to the outgoing link. If any relay fails, all downstream relays and clients will eventually expire and discard table entries associated with the group. In addition, the affected clients will re-construct the path to MR as described in Section 5.3.

MTor can also tolerate the failure of multicast tree root. If any client can not

connect to MR after she has tried a pre-configured number of different paths, she can simply select another active MR as described in Section 5.2. With high probability all participants will again connect to an unanimous MR to form a multicast tree across all group members. While in the rare scenario some clients may select different MRs (e.g. the original MR goes down for a short period of time before coming online again), the group can still recover from such inconsistency when it re-constructs multicast tree in the next session.

To reduce the group's vulnerability to slow relays, as well as deliberate DoS attacks by malicious relays which intentionally drop cells from upstream links, each heartbeat message provides a count of cells transmitted during the session, signed with the signing key of the MR (signing keys are specified in Tor descriptor documents). By comparing the received number of cells with the advertised count in the heartbeat message, the downstream clients can recognize such an attack and optionally re-connect to the MR via a different path.

5.6 Flow Control and Message Loss

MTor ensures only best-effort delivery of multicast messages. It uses TCP to disseminate messages reliably from parents to children in the multicast tree and for flow control.

In particular, when a multicast cell arrives at a relay, it is duplicated and enqueued on the internal output queues associated with each of the next hops. If an output queue reaches its capacity limit, incoming cells will be dropped on that queue; if all output queues reach their limit, then the relay blocks receiving from its previous hop. Due to potential message dropping at the application-layer, MTor offers best-effort, but potentially lossy multicast messaging (as do most multicast

schemes). The use of the predetermined bandwidth attribute in the group descriptor reduces the probability of loss, as relays are selected based on their ability to handle the group's predicted data rate.

Chapter 6

Evaluation

In this chapter we present the evaluation results of MTor in terms of its anonymity and network performance.

6.1 Network Performance

In this section, we measure the performance properties (i.e. bandwidth consumption, transmission time, computation overhead and probability of unreliability) of MTor and compare against unicast-based methods for group communication. The goals of our evaluation are three-fold: (1) measure the bandwidth saving achieved by MTor over unicast-based approach; (2) quantify the performance improvement as observed by group members; and (3) study the churn handling overhead and probability of unreliability due to relay failures.

6.1.1 Evaluation Methodology

We first describe the tools used to evaluate MTor. Specifically, we use TorPS [30], which allows us to measure MTor's bandwidth performance and anonymity as if it

were deployed on the live Tor network, and Shadow, which allows us to evaluate MTor's transmission time performance in an simulated network.

To provide an estimate of MTor's bandwidth consumption *on the actual Tor network*, we modified the Tor Path Simulator (TorPS) [30] to simulate MTor's tree construction algorithm over an one-month period of September 2014[1], using historical records of Tor consensus documents collected by the Tor Metrics Project [53]. During the simulation the multicast tree is re-constructed every hour. The bandwidth consumption is then derived from the average size of multicast trees. TorPS simulates the actual event of relays joining and leaving the Tor network using real relay and consensus data from Tor's historical records, and thus models the actual live Tor network as it existed at a specific past period in time. Using TorPS thus allows us to obtain an accurate estimate of MTor's performance had it been deployed on the live Tor network.

We also modified TorPS to estimate the probability of unreliability due to relay failure, as well as the resilience of Tor and MTor communication against traffic correlation attacks. In Section 6.1.7, we define the probability of unreliability and discuss MTor's churn handling performance. In Section 6.2, we adapt the security analysis techniques introduced by Jansen et al. [30] to measure the ability of a malicious adversary who controls some fraction of relays on the Tor network to de-anonymize group members.

To measure the network latency and transmission time as experienced by group members using MTor, we have implemented a prototype of MTor in C++ by adding approximately 1500 lines of code based on Tor version 0.2.3.25. We then emulated our prototype using Shadow [27] following a standard Tor network modeling approach [25]. Shadow is a discrete-event network simulator that runs actual Tor

[1]Using the September 2014 dataset, TorPS includes 6192 relays.

code using a synthetic network stack and a topological map of the live Tor network. Shadow allows us to simulate large-scale Tor deployments and measure performance for different application scenarios. Shadow has recently been used to evaluate Tor's circuit scheduling [27, 18] and congestion management algorithms [26], as well as its anonymity properties [30].

Because Shadow bypasses many OpenSSL encryption functions in order to allow researchers to track cells, we do not use authenticated group messages or end-to-end message encryption. The Shadow experiments assume public groups that anyone can join and send/receive messages. We separately evaluate the overheads of authenticated group messaging in MTor using micro-benchmarks.

6.1.2 Experimental setup

We use Shadow to simulate a Tor network of 455 relays, 1800 clients, and 500 client destinations (which we generically refer to below as "servers"). Relay capacities, geographic locations of relays, and link latencies between relays are configured according to the configuration supplied with Shadow, which itself is configured using data from Tor Metrics Portal [53] following Tor modeling best practices [25, 26]. Each server is assigned 100 MBps bandwidth and clients are assigned unlimited bandwidth, which is much higher than relay capacities and thus moves the performance bottleneck to the Tor network.

To model a loaded Tor network, we include 1800 clients that fetch files from any of the 500 servers via unicast Tor circuits. To match existing studies of behavior on the live Tor network [32], 1350 clients behave as interactive web clients that fetch files of 320KB in size, and sleep for up to one minute. Additionally, 300 clients repeatedly fetch 50KB, 1MB or 5MB files, sleeping one minute in between each fetch. These

types of clients continuously repeat a fetch-sleep cycle where they fetch files from a randomly selected server (out of 500 servers). Finally, another 150 clients behave as bulk clients (e.g., file sharers) that continuously fetch data from a random server and switch to a different server after every 5MB data transmission.

We include an additional 20 group communication clients in our Shadow topology. To support our baseline comparison, which we explain in more detail below, we also add one additional server that serves as external facilitator to support group communication via traditional unicast Tor circuits.

6.1.3 Performance Metrics

We evaluate the performance of MTor and Tor using four metrics: (1) the overall *network bandwidth* that is consumed to transmit the data to all clients; (2) the *transmission time*, which measures the time it takes for a receiver to receive the sender's complete message (time-to-last-byte); (3) the packet loss rate due to a mismatch between bandwidth capacity and the bandwidth requirements of real-time communication applications; and (4) the probability of unreliability due to relay failure.

Network bandwidth consumption is measured as the sum of bytes transmitted on each link in the Tor overlay network during the course of an experiment, which provides insight into the burden imposed on the Tor overlay network. The transmission time captures the latency experienced by end users, which tends to be dominated by bandwidth capacity for large messages. The packet loss rate estimates the packet loss due to unsatisfactory bandwidth capacity and network congestion. Finally, the probability of unreliability measures the impact of communication disruption due to relays on the multicast tree becoming unavailable during communication.

6.1.4 Modeling Group Communication Applications

To evaluate MTor's performance properties under different communication scenarios, we model three types of group communication applications. In all the MTor experiments, clients communicate directly via the multicast tree.

- **Single-source streaming.** In the single-source streaming application, a single non-anonymous server multicasts a file (e.g., representing a video or document) of 10MB to a group of 20 anonymous clients. In our baseline scenario, all clients connect to and receive data from the server via unicast Tor circuits. This scenario explores the transmission time improvement from using MTor in a typical initiator-responder scenario, where many initiators request the same data at around the same time.

- **Multi-source group streaming.** In the multi-source streaming application, we consider a group of 20 anonymous clients communicating with each other.

  When measuring the performance of MTor, the traffic is transmitted via the multicast tree. Since vanilla Tor does not support anonymous group communication, as our baseline for comparison, we consider a scenario in which all clients connect to an external service that "echoes" messages to all other connected clients. Tor clients connect to this external service, which we call the *facilitator*, through unicast Tor circuits. (This is effectively the strawman solution proposed in Section 5.4.)

- **Audio conferencing.** Our third use-case considers a group of 20 anonymous

clients doing real-time voice-over-IP communication. We assume VoIP is performed using Internet Low Bitrate Codec (iLBC) [19] at 1666 Bps, which is extremely robust to packet loss[2]. Again, for our baseline configuration, all clients that rely on vanilla Tor connect to a facilitator using dedicated circuits.

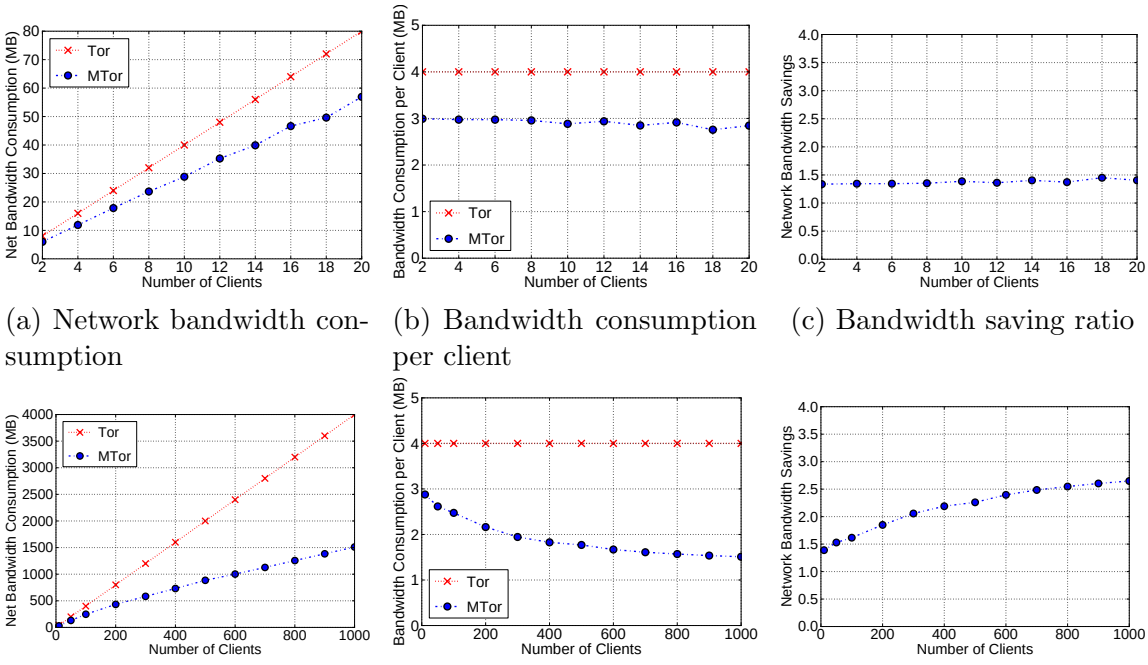
To model audio conferencing as a real-time application, each client queues a 1666-byte message per second for transmission to other clients. The old message is dropped if it is not sent before the new message gets queued. For both MTor and baseline experiments, we simulate the audio conferencing for 30 minutes to measure the message loss rate and transmission time distribution.

Limitation In our evaluation of group communication applications, we focus on the characteristics of the data transmission at the transport layer, such as overall network bandwidth consumption and transmission time distribution. To evaluate the impact of network congestion on real-time group communication applications, we additionally simulate the packet loss event in audio conferencing application, where old message is dropped if it is not sent before the new message gets queued.

It is important to note that we do not emulate an actual audio conferencing application – application layer behaviors such as iLBC codec, message ordering, retransmission etc. are omitted in our Shadow experiments. The absolute performance of audio conferencing application may vary when these implementation details are introduced. Nonetheless, we expect our evaluation results to be useful in understanding the relative performance advantage of MTor over unicast-based approaches. A realistic deployment and evaluation of these group communication applications over MTor on live Tor network is deferred to future work.

[2]iLBC is a mandatory standard for VoIP over Cable and is also used by Google Voice and Skype.

6.1.5 Impact on the Bandwidth Consumption



(a) Network bandwidth consumption

(b) Bandwidth consumption per client

(c) Bandwidth saving ratio

(d) Network bandwidth consumption

(e) Bandwidth consumption per client

(f) Bandwidth saving ratio

Figure 6.1: Network bandwidth consumption by MTor and the baseline approach via unicast Tor for small groups of up to 20 clients (top row) and large groups of up to 1000 clients (bottom row), for the multi-source group streaming application. We evaluate the bandwidth consumption with respect to 1MB worth of group messages that are collectively transmitted by the group's members. *(a,d)* The overall network bandwidth consumption for small and large groups. *(b,e)* The average network bandwidth consumed per client, for small and large groups. *(c,f)* The network bandwidth consumption ratio of MTor to Tor for small and large groups.

MTor offers the potential for significant bandwidth savings due to message de-duplication. To investigate how the Tor network could benefit from these savings (i.e., by having to forward less traffic), we focus in this section on the multi-source group streaming scenario. Recall that in the baseline setup, each client connects to an external service via unicast Tor circuits. We simulate data transmissions from each client to every other client in the group, and evaluate the resulting load on the Tor network as a function of the group's size. Our evaluation is based on paths produced

by our modified TorPS path simulator. We evaluate the bandwidth consumption when clients collectively transmit 1MB of messages to the group members (i.e., each client receives 1MB worth of message contents); as we show below, the overhead of sending 1MB to the group varies considerably between vanilla Tor and MTor.

Although we fix our experiments in this section on a 1MB-sized conversation, we remark that transmitting more data merely induces a linear increase in the amount of network bandwidth consumed for both MTor and baseline experiments.

Figure 6.1 shows the network bandwidth consumption that results from MTor and Tor as the number of clients increases from 1 to 1000. Tor's bandwidth consumption is derived theoretically as 4×client#×1MB, since 1MB data is transmitted along 3-hop Tor circuits to the facilitator for each client in the group. To measure MTor's bandwidth costs, we simulate tree construction 720 times and compute the average number of links in the resulting multicast trees; the bandwidth is then computed as the average number of links times 1MB. These figures demonstrate the bandwidth savings MTor could achieve for small (top row) and large (bottom row) group sizes.

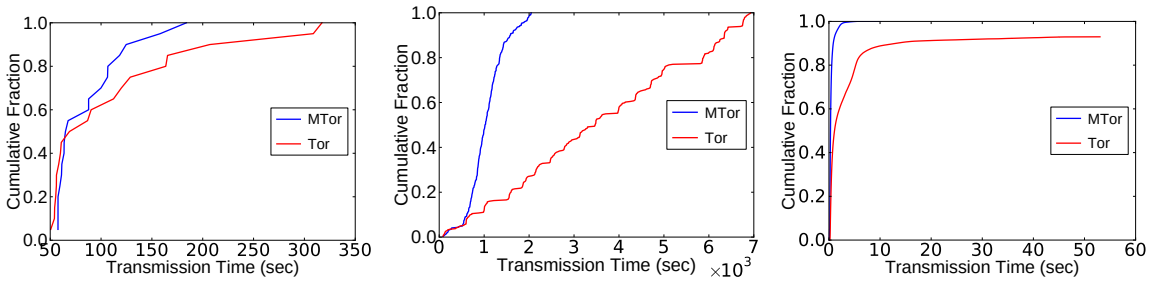
We make the following observations: bandwidth consumption in Tor increases linearly with the number of clients by a factor of 4, whereas in MTor bandwidth consumption is sublinear. The advantage of using MTor increases with group size; MTor reduces the bandwidth cost by approximately 62% over vanilla Tor for a large group with 1000 members (Figures 6.1a and 6.1d). The bandwidth savings in MTor is due to two factors: (i) in MTor, clients' paths are shorter (consisting of two hops from the client to the MR) since they do not include links from exits to the sender; and (ii) MTor removes unnecessary cell duplication when links are shared.

Figures 6.1b and 6.1e further highlight the benefits of cell de-duplication. Here, the figures plot the bandwidth that is consumed in the Tor network, averaged across the clients, as the size of the group increases. For Tor, each group member consumes

a fixed amount of 4MB bandwidth for each 1MB data transmitted, since no de-duplication occurs and each client receives the sender's communication via its own 3-hop Tor circuit. For MTor, when the size of group is 10, each client consumes on average 2.8MB bandwidth for each 1MB data transmitted. As the size of group increases to 1000, each client consumes on average only 1.5MB bandwidth, much closer to the theoretical lower bound of 1MB bandwidth necessary to serve a client. This is a direct result of de-duplication: links in the multicast tree are used by more than one client, providing the opportunity for bandwidth savings. As more clients join the group, these opportunities increase. For example, if a new client joins and its guard is already part of the group's multicast tree, then the only additional bandwidth cost due to that client is the cost of sending a copy of group message from the guard to the client.

Figures 6.1c and 6.1f plot the savings in network bandwidth consumption when MTor is used in place of Tor for group communication, and highlight our earlier results. MTor offers increasingly efficient group communication as the size of the group increases. The savings increase from 29% for a group of 10 members to 62% for a group of 1000 members.

Discussion We note that bandwidth saving in MTor is primarily due to three factors: (1) each MTor client requires at most 3 links to connect to MR, whereas Tor always requires a separate 4-link path for each connection; (2) as number of clients increases it becomes more likely that a new client picks a guard node that is already in the multicast tree, in which case it adds only one link in the overlay network; and (3) MTor avoids the use of external facilitator to forward data.



(a) (b) (c)

Figure 6.2: Cumulative distribution of transmission time (determined using Shadow) of *(a)* 10MB files from one sender to the group during single-source streaming, *(b)* 10MB files from each client to the group during multi-source streaming, and *(c)* 1666-byte message per second from each client to the group during audio conferencing.

6.1.6 Impact on Transmission Time

We next consider performance from the perspective of group members. Here, we emulate Tor and MTor in Shadow simulator. We use *transmission time* to capture the delay experienced by end users to receive a message, since it encompasses both network congestion as well as queuing delay at the sender, receiver, and the intermediate relays. In other words, transmission time is an intuitive notion of a user's experience, which captures the bandwidth capacity between sender and receivers.

Figure 6.2 compares the transmission time distribution offered by vanilla Tor (using our baseline configuration) and MTor for each of our applications. We remark that the performance improvement from using MTor is largely attributed to reduced network congestion in the Tor network, instead of avoiding performance bottlenecks at the server: although the server in the baseline setup handles one order of magnitude more traffic than clients, it is configured with 100 MBps bandwidth, much higher than the bandwidth of relays in the experiment.

Single-source streaming Figure 6.2a shows the cumulative distribution of transmission time to transmit a 10MB file from a single server to 20 anonymous clients. In both MTor and Tor experiments, 20 files are received and their time-to-last-byte are

measured. Our Shadow experiments show that MTor provides observably improved transmission time and a much shorter tail than Tor for carrying out single-source streaming.

For this small group of 20 clients, MTor reduces the median transmission time by 22% from 86.7 seconds to 67.6 seconds. In the 99th percentile, the time is reduced by 43% from 317.3 seconds to 183.9 seconds. Overall, MTor reduces the latency for 55% of clients.

Multi-source group streaming Figure 6.2b shows the cumulative distribution of transmission time to transmit 10MB file during anonymous group communication. Since each client sends a copy of file to every other 19 clients, in total 390 copies of 10MB files are received by clients.

We observe that MTor significantly improves transmission time over vanilla Tor in doing anonymous multi-source group communication. For a small group of 20 clients, MTor reduces the median transmission time by 41.5% from 2773 seconds to 1328 seconds. In the 99th percentile, the time is reduced by 55% from 5074 seconds to 2285 seconds. Overall, MTor reduces the latency for 55% of clients.

In the baseline experiment, for every message that it receives, the external facilitator must transmit 19 copies (via 19 circuits) to the other group members. MTor improves performance by (i) using message de-duplication, (ii) avoiding potentially congested exit relays, and (iii) eliminating the need to forward messages through facilitators.

Audio conferencing Figure 6.2c shows the cumulative distribution of transmission times for the real-time audio conferencing application. Each client in the group attempts to send a 1666-byte message per second. To deliver real-time audio messages in a timely fashion, clients favor newer "audio samples" and drop unsent messages if a new 1666-byte message is available.

We make the following observations: MTor successfully delivers 100% of the messages, while vanilla Tor delivers only 93% of all messages. At the 50 percentile, MTor reduces the transmission time by 73% from 1.1 to 0.3 seconds. The slowest message takes 2.5 seconds to be delivered in MTor, whereas it is 53 seconds in Tor. The result shows that MTor enables anonymous group communication with real-time delivery requirements, while Tor's message loss rate and long-tail distribution of transmission time would considerably reduce the user experience for these applications.

Discussion MTor is able to offer shorter transmission time because it imposes a much lower burden on the sender: in vanilla Tor, the sender must duplicate each outgoing message for each client's connection, since each client connects to the sender with its own anonymous path. In MTor, the sender only needs to send a single copy of each message, relying instead on the multicast tree to propagate it to the receivers. As the number of receivers grows, this asymmetry becomes more pronounced.

The encouraging results suggest that MTor can significantly improve transmission time over Tor for both small and large-sized group communication. Such improvement is particularly useful when anonymous real-time communication is desired.

6.1.7 Churn Handling Evaluation

In this section we evaluate the efficiency of the churn handling mechanism described in Section 5.5.

Suppose the heartbeat cell is sent by MR every h seconds, the timer expires after t seconds if not refreshed by heartbeat cell, and the construction of a new path from client to MR takes p seconds. If any relay fails, the downstream clients will be disconnected from MR for $t + p$ seconds, during which each client detects timer

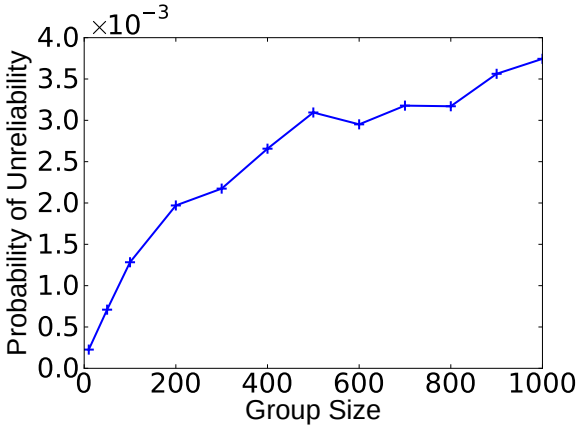


Figure 6.3: MTor's probability of unreliability due to relay failure

expiration and reconnects to MR via a 2-hop circuit. Since each cell has 512 bytes, the heartbeat cells will consume $512/t$ Bps of bandwidth per link.

To quantify the unreliability due to network churn, we define the probability of unreliability as the percentage of the time that *any* client in the group is disconnected from the MR. We note that this is a conservative measure of unreliability, since it assumes the disconnection of any client will impact all other clients in the group.

As part of our experimental setup, we assume that the heartbeat cell is sent every $h = 3$ seconds, and the timer expires after $t = 9$ seconds. As evaluated using the Torflow utility [39], the construction of a 3-hop path takes roughly $p = 6$ seconds. Under this setup, the heartbeat message consumes only 170 Bps bandwidth per link. Figure 6.3 shows the probability of unreliability for groups of size 10 to 1000, estimated via simulation in our TorPS variant over the one month period of September 2014. Recall that TorPS uses historical data from the live Tor network to simulate network churn. We remark that for a group of size 1000, the probability of unreliability is only 0.37% — that is, less than 15 seconds of communication will be disrupted during an hour-long communication session.

6.1.8 Authentication Microbenchmarks

Neither Shadow nor TorPS allows us to measure the computational overhead of the message authentication scheme described in Section 5.4, since neither simulator performs actual cryptographic operations. Instead, we next describe microbenchmarks that allow us to estimate the rate at which clients can generate signatures and relays can verify them.

We use OpenSSL version 1.0.1's benchmarking capability to measure the overhead of signing and verifying 283-bit ECDSA signatures, as well as the cost of computing SHA2 hashes. Our "client" runs a MacBook Pro with quad-core 2.2GHz Intel Core i7 processor and is able to generate 1604 signatures and 267K hashes per second. Measurements for our "relay" are taken from a commodity server with a quad-core 2.67GHz Xeon X3450 processor; the relay is able to verify 752 signatures per second and can perform 375K hashes per second. All measurements are pinned to a single core.

As described in Section 5.4, we can fit 91 hashes into a single signature cell when the hashes are truncated to 40-bits. Based on the measurements above, the client can send 95409 authenticated cells per second (equivalently 49 MBps). The amortized verification rate for the relay is 58507 cells per second; our relay is able to forward 30 MBps of authenticated group communication data, per dedicated core.

6.2 Anonymity Performance

Tor is known to be vulnerable against *traffic correlation* attacks in which an adversary who observes traffic entering and leaving the anonymity network can correlate that traffic to identify pairs of communicating parties. Prior work has shown that traffic correlation is an effective means of de-anonymizing Tor users, and can be performed

at low cost using statistical sampling techniques [35]. Arguably, it is the most serious threat against Tor users' anonymity [21, 55], as it directly exposes the identities of the communicating parties and can be carried out either by network operators or relay operators.

In this section, we evaluate how the use of MTor for group communication affects an adversary's ability to de-anonymize users through traffic correlation. The goal of this section is to validate that MTor performs comparably to Tor in anonymity.

6.2.1 Adversary Model and Goals

Rank	**Bandwidth (MBps)**	**Largest family member**
1	327	bolobolo1
2	207	torpidsUAitlas
3	190	PrivacyRepublic0019
4	189	orion
5	155	AccessNow14

Table 6.1: Tor *families* with top observed bandwidth on September 30th, 2014. The total observed bandwidth of all relays is 13 GB/s

We conservatively assume that an adversary is able to perfectly correlate traffic—i.e., if it observes Tor cells belonging to the same flow at two different points in the network, then the adversary can discern with perfect accuracy that those packets do indeed belong to the same Tor circuit. Hence, our results should be interpreted as a conservative measure of anonymity: real-world adversaries may not have the computational resources to perfectly correlate traffic flows.

Further, we assume an adversary that runs relays in the Tor network and uses these relays to observe traffic, correlate flows, and de-anonymize users. In particular, we provision the adversary with an *observed bandwidth budget* of 131MBps, 327MBps or 656MBps, which it may use to operate one or more relays in the Tor network,

such that the combined bandwidth of his relays does not exceed his bandwidth budget. The adversary must fix his selection of relays and is not allowed to change which relays it controls during the course of an experiment. We parameterize the adversary's bandwidth budget to consider MTor's security against adversaries of varying strength. As shown in Table 6.1, our bandwidth budgets conservatively model adversaries that have up to twice the observed bandwidth of the largest Tor families as of September 30th, 2014. (A Tor *family* consists of relays that report that they are administered by the same entity.) These bandwidth budgets respectively correspond to 1%, 2.5%, and 5% of the total observed bandwidth reported by all relays as of September 30th, 2014.

To carry out a traffic correlation attack in vanilla (unicast) Tor, the adversary needs to control both sides of a circuit (i.e., the guard and exit relay) to observe (and later correlate) the source and destination of communication.

We conservatively assume that the adversary's guard relay exhibits enough uptime to obtain the GUARD and STABLE flags. We additionally assume that the adversary's exit relay does not have the GUARD flag but does have an exit policy that allows communication to all addresses and ports; this increases its chance of being selected as an exit. All of the adversary's relays have sufficient bandwidth to obtain the FAST flag.

6.2.2 Anonymity Metrics

We consider an unicast connection as compromised if the adversary observes traffic at both ends (i.e. guard and exit relay in Tor) of the anonymous connection. The definition of compromise in a group communication setting is less clear since there

are potentially many receivers for a given message. In our anonymity evaluation, we consider two types of compromise w.r.t. attacks defined in Section 2.1:

- **Linkage.** We say two clients in the same communication group are *linked* (i.e., correlated) if the adversary observes each of their guard traffic. The adversary does not need to view their guard traffic simultaneously; observing their guard traffic even at different points during the group communication is sufficient to allow the adversary to determine that the two clients belong to the same communication group, since traffic belonging to the same group may be identified by inspecting the messages' GID.

- **Membership identification.** An adversary who controls a client's guard can determine whether that client is participating in a given group by examining the binding proofs (which contain a group's unique GID).

In our anonymity analysis, we conservatively assume that two given clients are *linked* if both guards are compromised at least once over the period of simulation; and a client is *identified as group member* if its guard is compromised. Notice that, by definition, *linkage* $\leq$ *membership identification* in terms of their probability of occurrence.

To measure susceptibility to traffic correlation attacks, we adopt the following security metrics from Johnson et al. [30] since we believe they are the most relevant to users of Tor:

- **Compromise rate:** the probability distribution on the fraction of paths that are compromised (w.r.t. linkage or membership identification) for a given user (in a given period); and

- **Time to first compromise:** the probability distribution on the time until the first path compromise (w.r.t. linkage or membership identification).

6.2.3 Experimental Setup

We envision that most users will continue to use vanilla Tor as their primary means of anonymous communication: that is, they will continue to use unicast communication to browse web, send emails, etc. Simultaneously, a smaller percentage of Tor users will use MTor to participate in group communication.

User Model For the unicast Tor users, we adopt the user models introduced by Johnson et al. [30] that are intended to reflect the behavior of actual users of the live Tor network. These user models consist of a sequence of Tor streams and the times at which they occur. Here, streams include DNS resolution requests in addition to TCP connections to specific destinations. Johnson et al. construct these models by using client applications on the live Tor network and tracing the behavior of the local Tor client. We use models consisting of Tor users who use *(i)* Gmail/Google Chat, *(ii)* Google Calendar/Docs, *(iii)* Facebook, and *(iv)* perform web searches.[3]

For MTor clients, we consider a large "webcasting" scenario in which 5000 MTor clients participate in the same group and receive multicast messages from a single sender. These webcasting sessions last for an hour, after which time the clients all leave the group and join a new group webcasting session with (w.h.p.) a new MR. This process repeats for the duration of the simulation.

Attacker configurations We first determine the bandwidth allocation between

[3]We remark that in MTor, all traffic is sent within the Tor network. Unlike vanilla Tor, MTor does not use exit policies since exit relays are not used. Consequently, selecting the relay path to the MR in MTor is not affected by the group members' choice of application—this is in contrast to standard unicast Tor where the choice of application (or more specifically, the destination port of egress traffic) influences relay selection, since a compatible exit relay must be chosen. This has an interesting effect on anonymity: unlike vanilla Tor, MTor's susceptibility to traffic correlation attacks is independent of its users' choice of application.

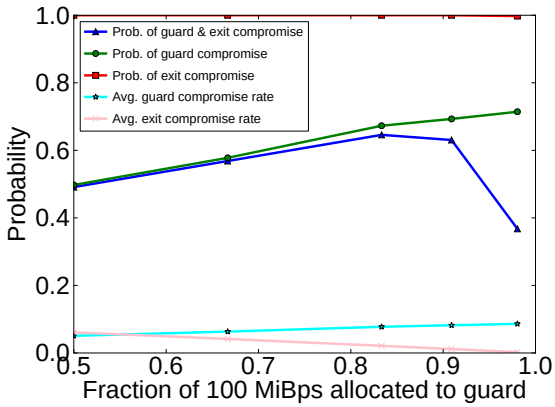


Figure 6.4: Probability of observing traffic (y-axis) for various bandwidth allocation strategies between the guard and exit (x-axis), using Tor consensus data from April 2014 through September 2014.

guard and exit relays that maximizes the adversary's ability to de-anonymize ordinary unicast Tor users. We tested guard-to-exit bandwidth ratios of 1:1, 2:1, 5:1, 10:1 and 50:1 using the TorPS path simulator. Figure 6.4 shows the compromise rate with varying bandwidth allocation ratios between guard and exit relays. A 5:1 ratio maximizes the probability of compromising both sides of at least one stream during the simulation period (blue line), which we adopt in the rest of this section. This confirms an earlier result by [30].

Since exit relays are not used by MTor, adversaries who attempt to de-anonymize group communication will spend their entire bandwidth budget in controlling guard relays. Recall that an adversary succeeds in linkage and membership identification correlation attacks by controlling the guard relay(s) used by a group's clients.

To assign selection weights to adversary relay given its controlled bandwidth, we use the fact that observed bandwidth that relays report in their consensus are correlated with their consensus weights. We use linear regressions on the relays in the consensus document during the simulation period to convert observed bandwidth of adversary's relays to consensus weight, where we use observed bandwidth as predictor

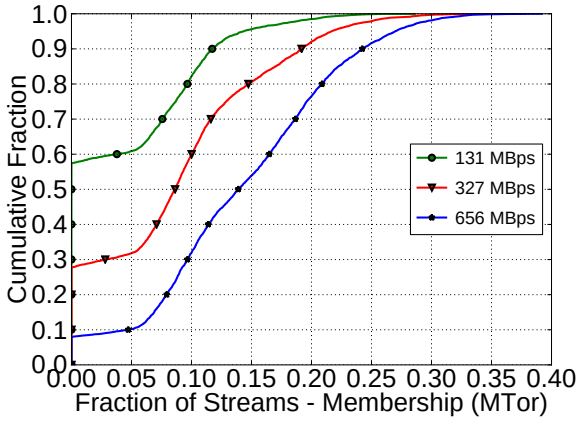


Figure 6.5: Cumulative distribution of the fraction of streams that allow the adversary to perform membership identification (i.e., the compromise rate for membership identification correlation attacks). The adversary's bandwidth budget is shown in the figures' legends.

and consensus bandwidth as descriptor. We use separate regressions for guard relays and exit relays, which result in correlations of determination of $r^2 = .55$ and $r^2 = .63$, respectively.

6.2.4 Evaluation Results

For both unicast Tor and MTor clients, we use TorPS to conduct 5000 Monte Carlo simulations of six months' client activity spanning the period from April 2014 to September 2014. We use the output of these simulations to evaluate the compromise rate and time to first compromise for Tor and MTor, for the linkage and membership identification attacks described above.

The adversary's ability to perform membership identification attack in MTor is depicted in Figure 6.5. The figure shows the cumulative distribution over the fraction of streams that an adversary is able to compromise (i.e., determine that the client is a member of the group). Our results indicate that an adversary who continuously contributes 131MBps of guard bandwidth to the network fails to identify more than

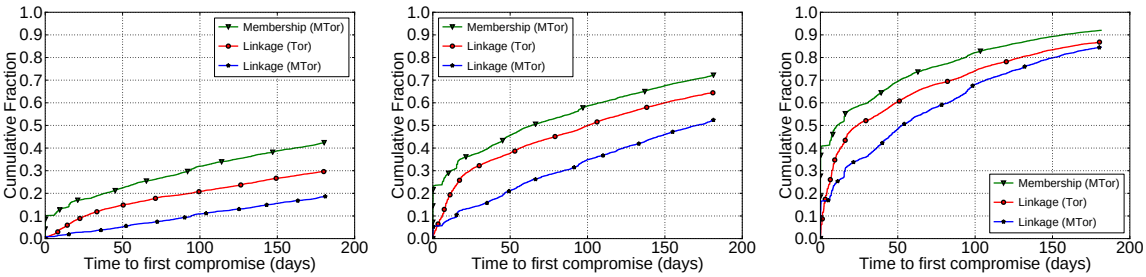


Figure 6.6: Time to first compromise with an adversary budget of *(a)* 131MBps, *(b)* 327MBps, and *(c)* 656MBps.

58% of the MTor clients during the simulation's six-month window. For 90% of the clients, the adversary is able to successfully determine group membership for only approximately 12% of the clients' multicast groups. (Recall that MTor clients change groups every hour.) MTor fares worse against more well-provisioned adversaries, although we note that even against an adversary who would constitute the largest contributor to Tor (the 327MBps case), 70% of clients have fewer than 12% of their streams compromised.

Figure 6.6 plots the cumulative distribution of the time to first compromise for MTor and Tor. A direct comparison between MTor and Tor is not possible, since the latter uses unicast workloads (web browsing, etc.) while the former is based on group communication. Generally, however, we expect MTor to provide *greater* resistance to linkage attacks than vanilla Tor for most clients: In vanilla Tor, exit relays are chosen independently for each new circuit, while the choice of guard relays persists across circuits.[4] An adversary who controls an exit relay can therefore wait until his relay is chosen. In contrast, MTor avoids the use of exit relays, requiring the adversary to control the guard relays of the two clients it is attempting to directly link. Adversaries who are not sufficiently lucky to operate the guards must

[4]The Tor Project is currently investigating how often Tor guard relays should be rotated [12, 14]. In the current version of Tor, a client rotates guards between 30 and 60 days (uniformly chosen).

wait potentially months before clients select other guards. This trend is observable in Figure 6.6 for all tested attacker strengths: albeit with different underlying workloads, the adversary is more quickly able to perform linkage correlation attacks against Tor than it is against MTor.

Against our 131MBps adversary, approximately 69% of clients were not identified as being a group member within 100 days. Against an adversary who would constitute the largest contributor to Tor (the 327MBps case), roughly 40% of clients were not identified in that same period. We note that these results should be considered conservative measures of the anonymity offered by MTor, since in practice, most users would presumably not continuously participate in a group for such a long duration.

Comparing Figures 6.6a, 6.6b, and 6.6c, we observe that the time-to-first compromise increases roughly linearly with the adversary's provisioned bandwidth budget, for both membership identification and linkage attacks. This is due to Tor's bandwidth-weighted relay selection policy: clients choose relays proportional to how much bandwidth they contribute to the network, thus increasing the adversary's bandwidth budget by a constant factor also increases the probability that clients will select its relays by roughly the same factor.

Chapter 7

Summary

In this chapter we summarize the dissertation and provide a few promising directions for future work.

7.1 Discussion

Incremental deployment MTor requires changes to both Tor clients and relays. Importantly, however, since MTor works alongside standard unicast Tor, it does not require that all clients and relays support anonymous multicast communication. A straightforward approach to incrementally deploying MTor involves the introduction of a new Tor flag, MTor, that is assigned to relays by the Tor directories if those relays support group communication. Once a sufficiently large number of relays advertise the MTor flag in their descriptors (hence offering diverse options for relay selection), MTor-capable clients can then choose amongst those relays when selecting and building a path to the MR.

Growth of the Tor network MTor offers bandwidth savings due in part to its de-duplication of messages. If the Tor network expands to include more relays with the

STABLE and FAST flags, then the probability that clients using MTor will select the same relays in their paths to the MR will decrease, thus providing fewer opportunities for de-duplication. One possible approach to counter this effect is to adopt the MTOR flag described above, and assign it only to a fixed number of relays such that the opportunities for de-duplication also remain fixed. An alternative approach is to bias the selection of the middle relay in MTor circuits by incorporating the GID into the relay selection process; here, the intended effect is to cause clients to select relays that are more likely already participating in the multicast tree.

Fortunately, our TorPS simulation using recent consensus data from the live Tor network indicates that opportunities for de-duplication *do* exist in current Tor (see Section 6.1.5). And, independent of de-duplication, MTor offers other bandwidth savings. Since it handles message distribution *within* the Tor network, MTor (i) eliminates the need to burden exit relays and, more importantly, (ii) reduces network bandwidth consumption by removing at least two hops between clients in the same group. The latter holds since in the worst case in MTor, traffic traverses a 2-hop path to the MR and a 2-hop path down to another group member; in contrast, a client using vanilla Tor and an external facilitator must send traffic via a 3-hop path to the facilitator, which then forwards the traffic via a 3-hop path to the client.

Adjusting guard rotation Recent work proposes replacing Tor's current guard design—which now consists of using three guards that are discarded after between 30 and 60 days of use—with a single fixed guard that is maintained for nine months [14]. The policy change directly targets Tor's susceptibility to traffic correlation attack by requiring the adversary to wait longer if it does not control a particular target user's guard relay; that is, it forces the adversary to get lucky early on. If adopted, such a policy will also significantly improve MTor's anonymity properties, since the

findings are directly applicable: a prerequisite of both linkage and membership identification attacks is that the adversary controls the user's guard relay, and a longer guard rotation period means that the adversary must wait longer for its relays to be chosen as guards. Fortunately, the Tor Project seems prone to move towards this longstanding, single-guard model [13].

7.2 Conclusion

This dissertation presents the design and implementation of MTor, which to the best of our knowledge is the first system that provides low-latency anonymous group communication with a decentralized trust infrastructure. MTor gracefully scales with the size of the communication group by constructing multicast trees on top of the Tor overlay network, and allows dynamic group composition without relying on global coordination.

We performed comprehensive analysis of MTor's bandwidth consumption, latency, unreliability, and anonymity performance using recently proposed simulation techniques with realistic models of the Tor topology and historical datasets of Tor relay information. Our results are encouraging: the bandwidth consumption and latency performance scale gracefully as additional clients join the group communication. We show that MTor achieves significant performance improvements that enable new forms of anonymous group communication (e.g., anonymous VoIP) while providing anonymity that is comparable to that provided by vanilla Tor.

7.3 Future Directions

Our long term goal is to integrate MTor into Tor ecosystem to enable scalable anonymous group communication for Tor's hundreds of thousands of daily users. The work presented in this dissertation is the first step towards this goal.

In this section, we discuss some open questions and future directions in order to make MTor into reality, or more generally, to enable anonymous group communication in adversarial environment.

7.3.1 Sybil Attack Mitigation

Multi-source anonymous group communication with dynamic membership presents a unique challenge to message de-multiplexing: it is infeasible to affix verifiable identity information of sender to messages, making it hard to aggregate messages securely per-source at receiver. On the other hand, to work in a potentially adversarial environment, it is necessary for group communication protocol to prevent Sybil attacks, in which misbehaving sender may create unlimited anonymous Sybil identities or spoof identities of other clients to disrupt group communication.

To enable *secure* message de-multiplexing in multi-source anonymous group communication, one possible solution is to de-multiplex messages based on the hash value of identities of relays each message has traversed. More specifically, each relay updates message's de-multiplex key by hashing it with its own identity before forwarding the message to its neighbors, such that the calculation of the de-multiplex key is effectively distributed across relays on the path from source to destination. As part of future work, we hope to verify that the solution does prevent Sybil attacks without introducing unexpected vulnerability for user's anonymity.

7.3.2 Denial-of-Service Attack Mitigation

While multicast primitive enables efficient group communication, it also opens opportunity for DoS (flooding) attack since it amplifies messages by design. To mitigate DoS attack, MTor provides authenticated multicast, where relays verify received messages' signature and drop messages that fail verification. However, such approach incurs undesirable computation overhead for each forwarded message even when there is no DoS attack.

Ideally, we would like to mitigate DoS attack in anonymous communication in such a way that (1) blocks attack traffic at the relay closest to the source to minimize its impact, and (2) incurs no computation or bandwidth overhead when there is no DoS attack.

One promising idea is to use Pushback [23] to dynamically push message authentication functionality from receiver to the relay closest to attacker when unauthenticated messages are detected, and turn off message authentication at intermediate relays if they have not seen unauthenticated messages for a pre-configured period of time. As part of future work, we plan to fulfill the design detail, implement it in MTor and verify its effectiveness against DoS attack.

7.3.3 Secure Congestion Control

While message authentication can prevent unauthenticated messages from impacting network, it is not useful against insider attack where attacker can also send authenticated messages. For example, some misbehaving clients may keep sending messages regardless of their allocated share of bandwidth. For MTor to work in an adversarial environment, we need a mechanism to *enforce* fair bandwidth allocation

among anonymous clients without requiring them to coordinate with each other or with any global authority.

However, the requirement of anonymity presents unique challenge to enforcing congestion control. Unlike non-anonymous multicast system, MTor can not track the amount of bandwidth consumed by clients. One promising solution is to push rate limit from multicast root to clients such that each relay enforces the rate limit it received from upstream relays. We leave the design and evaluation of secure congestion control to future work.

Appendix A

MTor Pseudocode

The implementation of MTor includes an addition of 1000 lines of C++ code based on Tor-0.2.3.25. In this chapter we provide code snippets to outline the implementation of MTor. Please refer to [34] for complete implementation and evaluation suit.

In file main.c:

```
1  /** Perform regular maintenance tasks. This function gets run once per
2   * second by second_elapsed_callback().
3   */
4  void run_scheduled_events()
5  {
6    ... /* Tor code */
7
8    /** increase channel package_window and deliver_window*/
9    channel_increase_window();
10 }
```

In file config.c:

```
1  static config_var_t _option_vars[] = {
2
3    ... /* configuration options from Tor */
4
5    /* default MTor port used by application */
6    VPORT(MulticastPort, LINELIST, 9050),
7
```

```
  /* default bandwidth limit for MTor application*/
  V(MulticastBandwidth, MEMUNIT, "5 MB"),
}
```

In file command.c:

```
/* Process a <b>cell</b> that was just received on <b>conn</b>. */
void
command_process_cell(cell_t *cell, or_connection_t *conn)
{
  ... /* Tor code */

  switch (cell->command) {
    case CELL_MULTICAST_BEGIN:
    case CELL_MULTICAST_HOLD:
    case CELL_MULTICAST_DATA:
      ++stats_n_multicast_cells_processed;
      command_process_multicast_cell(cell, conn);
      break;

    ... /* Tor code */
  }
}

/** Process a 'multicast_data' <b>cell</b> that just arrived from
 * <b>conn</b>.
 */
static void
command_process_multicast_cell(cell_t *cell, or_connection_t *conn)
{
  circuit_t *circ, *channel;
  circid_t origin_circ_id;
  edge_connection_t *edge_conn;

  /* Multicast this cell to all interfaces except the incoming one*/
  origin_circ_id = cell->circ_id;
  if (server_mode(get_options()))
    channel_multicast_cell(cell->channel_id, cell, origin_circ_id, conn);

  cell->circ_id = origin_circ_id;
  channel = channel_get_by_channelid(cell->channel_id);

  for (circ=channel; circ; circ=circ->next_multicast) {
    switch (cell->command) {
      case CELL_MULTICAST_DATA:
        if (CIRCUIT_IS_ORIGIN(circ)) {
          circuit_receive_multicast_data(cell, circ);
        }
        break;
      case CELL_MULTICAST_BEGIN:
        channel_set_state(cell->channel_id, CHANNEL_STATE_OPEN);
        if (CIRCUIT_IS_ORIGIN(circ)) {
```

```
47            /* Like if we have received cell_created */
48            if (circ->state == CIRCUIT_STATE_BUILDING) {
49              origin_circuit_t *origin_circ = TO_ORIGIN_CIRCUIT(circ);
50              circuit_receive_multicast_begin(origin_circ);
51            }
52            /* Like if we have received relay_command_begin */
53            for (edge_conn = TO_ORIGIN_CIRCUIT(circ)->p_streams; edge_conn;
54                 edge_conn = edge_conn->next_stream) {
55              entry_connection_t *entry_conn = EDGE_TO_ENTRY_CONN(edge_conn);
56              if (entry_conn->channel_id != cell->channel_id)
57                continue;
58              edge_conn->_base.state = AP_CONN_STATE_OPEN;
59              /* handle anything that might have queued */
60              if (connection_edge_package_raw_inbuf(edge_conn, 1, NULL) < 0) {
61                /* (We already sent an end cell if possible) */
62                connection_mark_for_close(TO_CONN(edge_conn));
63                continue;
64              }
65            }
66          }
67          break;
68        case CELL_MULTICAST_HOLD:
69          channel_set_state(cell->channel_id, CHANNEL_STATE_HOLD);
70          break;
71      }
72    }
73  }
```

In file or.h:

```
1  /** Type for sockets listening for Multicast requests*/
2  #define CONN_TYPE_AP_MULTICAST_LISTENER 16
3
4  /* A Multicast SOCKS proxy connection from the user application to
5   * the onion proxy. */
6  #define CONN_TYPE_AP_MULTICAST 17
7
8  /** The circuit is used for Tor Multicast. */
9  #define CIRCUIT_PURPOSE_MULTICAST 20
10
11 /* Types for channel states */
12 #define CHANNEL_STATE_NONE 0
13 #define CHANNEL_STATE_BUILDING 1
14 #define CHANNEL_STATE_HOLD 2
15 #define CHANNEL_STATE_OPEN 3
16
17 /* Types for multicast cell */
18 #define CELL_MULTICAST_BEGIN 100
19 #define CELL_MULTICAST_HOLD 101
20 #define CELL_MULTICAST_DATA 102
21
22 typedef struct cell_t {
```

```
23    ... /* original cell_t fields */
24
25    /** Identify a multicast channel*/
26    channelid_t channel_id;
27  };
28
29  typedef struct entry_connection_t {
30    ... /* original cell_t fields */
31
32    /** Identify a multicast channel*/
33    channelid_t channel_id;
34  };
35
36  typedef struct circuit_t {
37    ... /* original cell_t fields */
38
39    channelid_t channel_id;
40    uint8_t channel_state;
41
42    /* Next circuit in linked list of all circuits
43     * with the same channel_id. */
44    struct circuit_t *next_multicast;
45
46    /* Next circuit in linked list of circuits
47     * with the different channel_id. */
48    struct circuit_t *next_channel;
49  };
50
51  typedef struct or_options_t {
52    ... /* original cell_t fields */
53
54    /* How much bandwidth, on average, are we willing
55     * to use for multicast connection in a second? */
56    uint64_t MulticastBandwidth;
57
58    /* Ports to listen on for Multicast SOCKS connections. */
59    config_line_t *MulticastPort_lines;
60  };
```

In file circuituse.c:

```
1   /** Find an open circ that we're happy to use for <b>conn</b> and return 1. If
2    * there isn't one, and there isn't one on the way, launch one and return
3    * 0. If it will never work, return -1.
4    */
5   static int
6   circuit_get_open_circ_or_launch(entry_connection_t *conn,
7                                   uint8_t desired_circuit_purpose,
8                                   origin_circuit_t **circp)
9   {
10    ... /* original Tor code */
11
```

```
12    /* Add newly created circuit to associated channel */
13    if (circ && conn->channel_id > 0) {
14      TO_CIRCUIT(circ)->channel_id = conn->channel_id;
15      channel_search_and_append(conn->channel_id,
16                                TO_CIRCUIT(circ));
17    }
18    ... /* original Tor code */
19  }
```

In file connection_or.c:

```
1   /* Pack the cell_t host-order structure <b>src</b> into network-order
2    * in the buffer <b>dest</b>.
3    */
4   void
5   cell_pack(packed_cell_t *dst, const cell_t *src)
6   {
7     char *dest = dst->body;
8     set_uint16(dest, htons(src->circ_id));
9     *(uint8_t*)(dest+2) = src->command;
10
11    /* Add field channel_id */
12    set_uint32(dest+3, htonl(src->channel_id));
13    memcpy(dest+7, src->payload, CELL_PAYLOAD_SIZE);
14  }
15
16  /* Unpack the network-order buffer <b>src</b> into a host-order
17   * cell_t structure <b>dest</b>.
18   */
19  static void
20  cell_unpack(cell_t *dest, const char *src)
21  {
22    dest->circ_id = ntohs(get_uint16(src));
23    dest->command = *(uint8_t*)(src+2);
24
25    /* Add field channel_id */
26    dest->channel_id = ntohl(get_uint32(src+3));
27    memcpy(dest->payload, src+7, CELL_PAYLOAD_SIZE);
28  }
```

In file circuitlist.c:

```
1   /* Allocate a new or_circuit_t, connected to <b>p_conn</b> as
2    * <b>p_circ_id</b>. If <b>p_conn</b> is NULL, the circuit is unattached. */
3   or_circuit_t *
4   or_circuit_new(channelid_t channel_id,
5                  circid_t p_circ_id, or_connection_t *p_conn)
6   {
7     ... /* original Tor code */
8
9     /* Add newly created circuit to its associated channel */
10    if (channel_id > 0) {
```

```
11      TO_CIRCUIT(circ)->channel_id = channel_id;
12      channel_search_and_append(channel_id, TO_CIRCUIT(circ));
13    }
14    ... /* original Tor code */
15  }
16
17  /* Increase delivery window and package window for all channels**/
18  void
19  channel_increase_window()
20  {
21    if (global_channellist == NULL)
22      return;
23
24    const or_options_t *options = get_options();
25    uint64_t multicast_bandwidth = options->MulticastBandwidth;
26    int multicast_window_increment = multicast_bandwidth / CELL_NETWORK_SIZE;
27    int multicast_window_max = 10*multicast_window_increment;
28
29    circuit_t *head, *circ;
30    edge_connection_t *edge_conn;
31    for (head=global_channellist; head; head=head->next_channel) {
32      for (circ=head; circ; circ=circ->next_multicast) {
33        if (!circ->marked_for_close &&
34            circ->purpose == CIRCUIT_PURPOSE_MULTICAST) {
35          /* Increase circuit window */
36          circ->package_window += multicast_window_increment*2;
37          circ->deliver_window += multicast_window_increment *2;
38          if (circ->package_window > multicast_window_max *2)
39            circ->package_window = multicast_window_max *2;
40          if (circ->deliver_window > multicast_window_max *2)
41            circ->deliver_window = multicast_window_max *2;
42
43          /* Increase stream window */
44          for (edge_conn = TO_ORIGIN_CIRCUIT(circ)->p_streams;
45               edge_conn; edge_conn = edge_conn->next_stream) {
46            edge_conn->package_window += multicast_window_increment;
47            edge_conn->deliver_window += multicast_window_increment;
48            if (edge_conn->package_window > multicast_window_max)
49              edge_conn->package_window = multicast_window_max;
50            if (edge_conn->deliver_window > multicast_window_max)
51              edge_conn->deliver_window = multicast_window_max;
52          }
53          /* Start reading from edge as if we received sendme cell */
54          circuit_resume_edge_reading(circ, NULL);
55        }
56      }
57    }
58  }
59
60  /* Get channel by <b>channel_id</b>. */
61  circuit_t *
62  channel_get_by_channelid(channelid_t channel_id)
```

```
63 {
64   circuit_t *circ;
65   for (circ=global_channellist; circ; circ = circ->next_channel) {
66     if (circ->channel_id == channel_id) {
67       return circ;
68     }
69   }
70   return NULL;
71 }
72
73 /* Append <b>next</b> to global_channellist. Return 1 if there
74  * exists circ with the same channel ID. */
75 int
76 channel_search_and_append(channelid_t channel_id, circuit_t *next)
77 {
78   tor_assert(channel_id > 0);
79   circuit_t *circ;
80   for (circ=global_channellist; circ; circ = circ->next_channel) {
81     if (!CIRCUIT_IS_ORIGIN(circ) &&
82         circ->channel_id == channel_id) {
83       break;
84     }
85   }
86   if (circ == NULL) {
87     next->next_channel = global_channellist;
88     global_channellist = next;
89     next->channel_state = CHANNEL_STATE_BUILDING;
90     return 0;
91   }
92   else {
93     next->next_multicast = circ->next_multicast;
94     circ->next_multicast = next;
95     next->channel_state = circ->channel_state == CHANNEL_STATE_OPEN?
96                           CHANNEL_STATE_OPEN : CHANNEL_STATE_HOLD;
97   }
98   return 0;
99 }
```

In file relay.c:

```
1  /* Deliver the cell to edge connections associated with the channel */
2  int
3  circuit_receive_multicast_data(cell_t *cell, circuit_t *circ)
4  {
5    int reason;
6    edge_connection_t *edge_conn;
7    for (edge_conn = TO_ORIGIN_CIRCUIT(circ)->p_streams; edge_conn;
8         edge_conn = edge_conn->next_stream) {
9      entry_connection_t *entry_conn = EDGE_TO_ENTRY_CONN(edge_conn);
10     if (entry_conn->channel_id != cell->channel_id)
11       continue;
12     connection_edge_process_relay_cell(cell, circ, edge_conn);
```

```
13    }
14    return 0;
15  }
16
17  /* Multicast cell to all circuits which have the <b>channel_id</b> */
18  int
19  channel_multicast_cell(channelid_t channel_id,
20                         cell_t *cell, circid_t circid,
21                         or_connection_t *conn)
22  {
23    circuit_t *circ, *channel;
24    or_connection_t *or_conn=NULL;
25    cell_direction_t cell_direction;
26
27    channel = channel_get_by_channelid(channel_id);
28
29    for (circ=channel; circ; circ=circ->next_multicast) {
30      if (circ->marked_for_close) {
31        tor_fragile_assert();
32        continue;
33      }
34      if (cell->command == CELL_MULTICAST_HOLD &&
35          circ->channel_state == CHANNEL_STATE_HOLD)
36        continue;
37      if (circ->n_circ_id) {
38        cell->circ_id = circ->n_circ_id;
39        or_conn = circ->n_conn;
40        cell_direction = CELL_DIRECTION_OUT;
41        if (or_conn != conn || cell->circ_id != circid) {
42          append_cell_to_circuit_queue(circ, or_conn, cell,
43                                       cell_direction, 0);
44        }
45      }
46      if (!CIRCUIT_IS_ORIGIN(circ) &&
47           TO_OR_CIRCUIT(circ)->p_circ_id) {
48        cell->circ_id = TO_OR_CIRCUIT(circ)->p_circ_id;
49        or_conn = TO_OR_CIRCUIT(circ)->p_conn;
50        cell_direction = CELL_DIRECTION_IN;
51        if (or_conn != conn || cell->circ_id != circid) {
52          append_cell_to_circuit_queue(circ, or_conn, cell,
53                                       cell_direction, 0);
54        }
55      }
56    }
57    return 0;
58  }
59
60  /* Create and multicast a cell with specified commands in header fields */
61  int
62  channel_multicast_command(channelid_t channel_id, uint8_t cell_command,
63                            uint8_t relay_command, circid_t circid,
64                            or_connection_t *conn, const char *payload,
```

```
65                                size_t payload_len)
66 {
67   cell_t cell;
68   relay_header_t rh;
69   circuit_t *circ, *channel;
70   or_connection_t *or_conn=NULL;
71   cell_direction_t cell_direction;
72
73   memset(&cell, 0, sizeof(cell_t));
74   cell.command = cell_command;
75   cell.channel_id = channel_id;
76
77   memset(&rh, 0, sizeof(rh));
78   rh.command = relay_command;
79   rh.stream_id = 0;
80   rh.length = payload_len;
81   relay_header_pack(cell.payload, &rh);
82   if (payload_len)
83     memcpy(cell.payload+RELAY_HEADER_SIZE, payload, payload_len);
84   return channel_multicast_cell(channel_id, &cell, circid, conn);
85 }
86
87
88 /** If <b>conn</b> has an entire relay payload of bytes on its inbuf (or
89  * <b>package_partial</b> is true), and the appropriate package windows aren't
90  * empty, grab a cell and send it down the circuit.
91  *
92  * Return -1 (and send a RELAY_COMMAND_END cell if necessary) if conn should
93  * be marked for close, else return 0.
94  */
95 int
96 connection_edge_package_raw_inbuf(edge_connection_t *conn, int package_partial,
97                                   int *max_cells)
98 {
99   ... /* original Tor code */
100
101   /* Call channel_multicast_command instead if it is a multicast circuits */
102   if (circ->channel_id > 0 && conn->_base.state == AP_CONN_STATE_OPEN) {
103     channel_multicast_command(circ->channel_id, CELL_MULTICAST_DATA,
104                               RELAY_COMMAND_DATA, 0, NULL,
105                               payload, length)
106   }
107   else {
108     connection_edge_send_command(conn, RELAY_COMMAND_DATA,
109                                  payload, length)
110   }
111   ... /* original Tor code */
112 }
113
114
115 /** Check if the package window for <b>circ</b> is empty (at
116  * hop <b>layer_hint</b> if it's defined).
```

```
117  *
118  * If yes, tell edge streams to stop reading and return 1.
119  * Else return 0.
120  */
121 static int
122 circuit_consider_stop_edge_reading(circuit_t *circ, crypt_path_t *layer_hint)
123 {
124   /* Stop reading all circuits with the same channel_id
125    * if it is a multicast circuit */
126   if (circ->channel_id > 0) {
127     if (circ->package_window <= 0) {
128       for (conn = TO_ORIGIN_CIRCUIT(circ)->p_streams; conn;
129            conn=conn->next_stream)
130         connection_stop_reading(TO_CONN(conn));
131       return 1;
132     }
133     return 0;
134   }
135   ... /* original Tor code */
136 }
137
138 static int
139 set_channel_blocked_on_circ(channelid_t channel_id, int block) {
140   circuit_t *channel, *circ;
141   edge_connection_t *edge = NULL;
142   channel = channel_get_by_channelid(channel_id);
143
144   if (block == 0) {
145     for (circ=channel; circ; circ=circ->next_multicast) {
146       if (circ->n_circ_id) {
147         if (circ->n_conn_cells.n > CELL_QUEUE_LOWWATER_SIZE) {
148           // can not unblock this channel
149           return 0;
150         }
151       }
152
153       if (!CIRCUIT_IS_ORIGIN(circ) && TO_OR_CIRCUIT(circ)->p_circ_id) {
154         or_circuit_t *orcirc = TO_OR_CIRCUIT(circ);
155         if (orcirc->p_conn_cells.n > CELL_QUEUE_LOWWATER_SIZE) {
156           // can not unblock this channel
157           return 0;
158         }
159       }
160     }
161   }
162
163   for (circ=channel; circ; circ=circ->next_multicast) {
164     if (circ->marked_for_close) {
165       tor_fragile_assert();
166       continue;
167     }
168     circ->streams_blocked_on_channel = block;
```

```
169
170     if (CIRCUIT_IS_ORIGIN(circ)) {
171       edge = TO_ORIGIN_CIRCUIT(circ)->p_streams;
172       for (; edge; edge = edge->next_stream) {
173         connection_t *conn = TO_CONN(edge);
174         edge->edge_blocked_on_circ = block;
175
176         if (block) {
177           if (connection_is_reading(conn))
178             connection_stop_reading(conn);
179         } else {
180           if (!connection_is_reading(conn))
181             connection_start_reading(conn);
182         }
183       }
184     }
185   }
186
187   return 0;
188 }
189
190 static int
191 set_streams_blocked_on_circ(circuit_t *circ, or_connection_t *orconn,
192                             int block, streamid_t stream_id)
193 {
194   if (circ->channel_id > 0) {
195     return set_channel_blocked_on_circ(circ->channel_id, block);
196   }
197   ... /* original Tor code */
198 }
```

In file connection_ap_multicast.c:

```
1  /* Process new bytes that have arrived on conn-\>inbuf. */
2  int
3  connection_multicast_process_inbuf(edge_connection_t *conn, int package_partial)
4  {
5    switch (conn->_base.state) {
6      case AP_CONN_STATE_SOCKS_WAIT:
7        if (connection_multicast_handshake_process_socks(EDGE_TO_ENTRY_CONN(conn)) <0) {
8          return -1;
9        }
10       return 0;
11     case AP_CONN_STATE_OPEN:
12       if (connection_edge_package_raw_inbuf(conn, package_partial, NULL) < 0) {
13         connection_mark_for_close(TO_CONN(conn));
14         return -1;
15       }
16       return 0;
17   }
18   tor_fragile_assert();
19   return -1;
```

```
20 }
21
22 /* Read another step of the socks handshake out of conn->inbuf. */
23 static int
24 connection_multicast_handshake_process_socks(entry_connection_t *conn)
25 {
26   socks_request_t *socks;
27   int sockshere;
28   const or_options_t *options = get_options();
29   int had_reply = 0;
30   connection_t *base_conn = ENTRY_TO_CONN(conn);
31
32   socks = conn->socks_request;
33   sockshere = fetch_from_buf_socks(base_conn->inbuf, socks,
34                                    options->TestSocks, options->SafeSocks);
35
36   if (socks->replylen) {
37     had_reply = 1;
38     connection_write_to_buf((const char*)socks->reply, socks->replylen,
39                             base_conn);
40     socks->replylen = 0;
41     if (sockshere == -1) {
42       /* An invalid request just got a reply, no additional
43        * one is necessary. */
44       socks->has_finished = 1;
45     }
46   }
47   return connection_multicast_handshake_rewrite_and_attach(conn);
48 }
49
50
51 /* Locate the multicast root for the group and connect to it via a circuit */
52 int
53 connection_multicast_handshake_rewrite_and_attach(entry_connection_t *conn)
54 {
55   socks_request_t *socks = conn->socks_request;
56
57   /* Find multicast root given user-specified group ID*/
58   const node_t *node = locate_rendezvous_point(socks->gid);
59   conn->chosen_exit_name = tor_strdup(hex_str(node->identity, DIGEST_LEN));
60   conn->channel_id = getNextChannelId();
61   conn->want_onehop = 0;
62
63   /* Construct a circuits connecting to multicast root */
64   return connection_ap_handshake_attach_circuit(conn);
65 }
```

Bibliography

[1] Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. LASTor: A Low-Latency AS-Aware Tor Client. In *IEEE Symposium on Security and Privacy (Oakland)*, 2012.

[2] M. AlSabah, K. Bauer, I. Goldberg, D. Grunwald, D. McCoy, S. Savage, and G. Voelker. DefenestraTor: Throwing out Windows in Tor. In *Privacy Enhancing Technologies Symposium (PETS)*, 2011.

[3] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller. Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492, IETF, 2006.

[4] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. Touching from a Distance: Website Fingerprinting Attacks and Defenses. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[5] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony I. T. Rowstron. Scribe: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. 20(8), October 2002.

[6] David Chaum. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[7] David L. Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[8] Yang-hua Chu, Sanjay G Rao, and Hui Zhang. A Case for End System Multicast. In *ACM SIGMETRICS Performance Evaluation Review*, 2000.

[9] Henry Corrigan-Gibbs and Bryan Ford. Dissent: Accountable Anonymous Group Messaging. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[10] Henry Corrigan-Gibbs, David Isaac Wolinsky, and Bryan Ford. Proactively Accountable Anonymous Messaging in Verdict. In *USENIX Security Symposium (USENIX)*, 2013.

[11] Stephen E Deering and David R Cheriton. Multicast routing in datagram internetworks and extended lans. *ACM Transactions on Computer Systems (TOCS)*, 8(2):85–110, 1990.

[12] Roger Dingledine. Research Problem: Better Guard Rotation Parameters, August 2011. Available at `https://blog.torproject.org/blog/research-problem-better-guard-rotation-parameters`.

[13] Roger Dingledine. Improving Tor's Anonymity by Changing Guard Parameters (blog post), October 2013. Available at `https://blog.torproject.org/blog/improving-tors-anonymity-changing-guard-parameters`.

[14] Roger Dingledine, Nicholas Hopper, George Kadianakis, and Nick Mathewson. One Fast Guard for Life (or 9 months). In *Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[15] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium (USENIX)*, August 2004.

[16] Roger Dingledine and Steven Murdoch. Performance Improvements on Tor, or, Why Tor is Slow and What We're Going to Do About It. `https://svn.torproject.org/svn/projects/roadmaps/2009-03-11-performance.pdf`, March 2009.

[17] Paul Francis. Yoid: Extending the Internet Multicast Architecture, 2000. Unpublished manuscript, available at `https://mpi-sws.org/~francis/yoidArch.pdf`.

[18] John Geddes, Rob Jansen, and Nicholas Hopper. IMUX: Managing Tor Connections from Two to Infinity, and Beyond. In *Workshop on Privacy in the Electronic Society (WPES)*, 2014.

[19] `http://tools.ietf.org/html/rfc3951`.

[20] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[21] Angèle Hamel, Jean-Charles Grégoire, and Ian Goldberg. The Mis-entropists: New Approaches to Measures in Tor. Technical Report 2011-18, Cheriton School of Computer Science, University of Waterloo, 2011.

[22] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. Website Fingerprinting: Attacking Popular Privacy Enhancing Technologies with the Multinomial Naive-bayes Classifier. In *ACM Workshop on Cloud Computing Security (CCSW)*, 2009.

[23] John Ioannidis and Steven M. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *Network and Distributed System Security Symposium (NDSS)*, 2002.

[24] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Symposium on Operating System Design & Implementation (OSDI)*, 2000.

[25] Rob Jansen, Kevin S Bauer, Nicholas Hopper, and Roger Dingledine. Methodically modeling the tor network. In *CSET*, 2012.

[26] Rob Jansen, John Geddes, Chris Wacek, Micah Sherr, and Paul Syverson. Never Been KIST: Tor's Congestion Management Blossoms with Kernel-Informed Socket Transport. In *USENIX Security Symposium (USENIX)*, August 2014.

[27] Rob Jansen and Nicholas Hopper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *Network and Distributed System Security Symposium (NDSS)*, 2012.

[28] Rob Jansen, Nicholas Hopper, and Yongdae Kim. Recruiting New Tor Relays with BRAIDS. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[29] Rob Jansen, Aaron Johnson, and Paul F Syverson. LIRA: Lightweight Incentivized Routing for Anonymity. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[30] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users Get Routed: Traffic Correlation on Tor By Realistic Adversaries. In *ACM Conference on Computer and Communications Security (CCS)*, November 2013.

[31] Seth F Kreimer. Technologies of protest: Insurgent social movements and the first amendment in the era of the internet. *University of Pennsylvania Law Review*, pages 119–171, 2001.

[32] Damon McCoy, Kevin Bauer, Dirk Grunwald, Tadayoshi Kohno, and Douglas Sicker. Shining Light in Dark Places: Understanding the Tor Network. In *Privacy Enhancing Technologies Symposium (PETS)*, 2008.

[33] Brad Moore, Chris Wacek, and Micah Sherr. Exploring the Potential Benefits of Expanded Rate Limiting in Tor: Slow and Steady Wins the Race with Tortoise. In *Annual Computer Security Applications Conference (ACSAC)*, December 2011.

[34] MTor Development Repository. `https://github.com/multicastTor`.

[35] Steven J. Murdoch and George Danezis. Low-Cost Traffic Analysis of Tor. In *IEEE Symposium on Security and Privacy (Oakland)*, 2005.

[36] Tsuen-Wan "Johnny" Ngan, Roger Dingledine, and Dan Wallach. Building Incentives into Tor. In *Financial Cryptography and Data Security*, 2010.

[37] Andriy Panchenko, Lukas Niessen, Andreas Zinnen, and Thomas Engel. Website Fingerprinting in Onion Routing Based Anonymization Networks. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2011.

[38] Ginger Perng, Michael K Reiter, and Chenxi Wang. M2: Multicasting Mixes for Efficient and Anonymous Communication. In *International Conference on Distributed Computing Systems (ICDCS)*, 2006.

[39] Mike Perry. Torflow: Tor network analysis. *Proc. 2nd HotPETs*, pages 1–14, 2009.

[40] Joel Reardon and Ian Goldberg. Improving Tor using a TCP-over-DTLS Tunnel. In *USENIX Security Symposium (USENIX)*, 2009.

[41] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for Web Transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[42] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to Leak a Secret. In *Advances in Cryptology (ASIACRYPT)*, 2001.

[43] `http://www.rooms.me`.

[44] `https://www.secret.ly`.

[45] Micah Sherr, Boon Thau Loo, and Matt Blaze. Towards Application-Aware Anonymous Routing. In *USENIX Workshop on Hot Topics in Security (HotSec)*, August 2007.

[46] Micah Sherr, Andrew Mao, William R. Marczak, Wenchao Zhou, Boon Thau Loo, and Matt Blaze. A3: An Extensible Platform for Application-Aware

Anonymity. In *Network and Distributed System Security Symposium (NDSS)*, 2010.

[47] Clay Shields and Brian Neil Levine. Hordes: A Multicast Based Protocol for Anonymity. *Journal of Computer Security*, 10(3):213–240, 2002.

[48] Robin Snader and Nikita Borisov. A Tune-up for Tor: Improving Security and Performance in the Tor Network. In *Network and Distributed System Security Symposium (NDSS)*, 2008.

[49] Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous Connections and Onion Routing. In *IEEE Symposium on Security and Privacy (Oakland)*, 1997.

[50] Can Tang and Ian Goldberg. An Improved Algorithm for Tor Circuit Scheduling. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.

[51] Al Teich, Mark S Frankel, Rob Kling, and Ya-ching Lee. Anonymous communication policies for the internet: Results and recommendations of the aaas conference. *The Information Society*, 15(2):71–77, 1999.

[52] `http://www.theguardian.com/world/2014/oct/16/-sp-revealed-whisper-app-tracking-users`.

[53] Tor Project, Inc. Tor Metrics Portal. `https://metrics.torproject.org/`.

[54] Tor Project, Inc. A Critique of Website Traffic Fingerprinting Attacks, 2014. Available at `https://blog.torproject.org/blog/critique-website-traffic-fingerprinting-attacks`.

[55] Tor Project, Inc. Tor FAQ: What Attacks Remain Against Onion Routing, 2014. Available at https://www.torproject.org/docs/faq.html.en#AttacksOnOnionRouting.

[56] Tor Project, Inc. Tor Rendezvous Specification, 2014. Available at https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=rend-spec.txt.

[57] Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware Path Selection for Tor. In *Financial Cryptography and Data Security (FC)*, 2012.

[58] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: A Camouflage Proxy for the Tor Anonymity System. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[59] http://www.whisper.sh.

[60] http://www.yikyakapp.com.