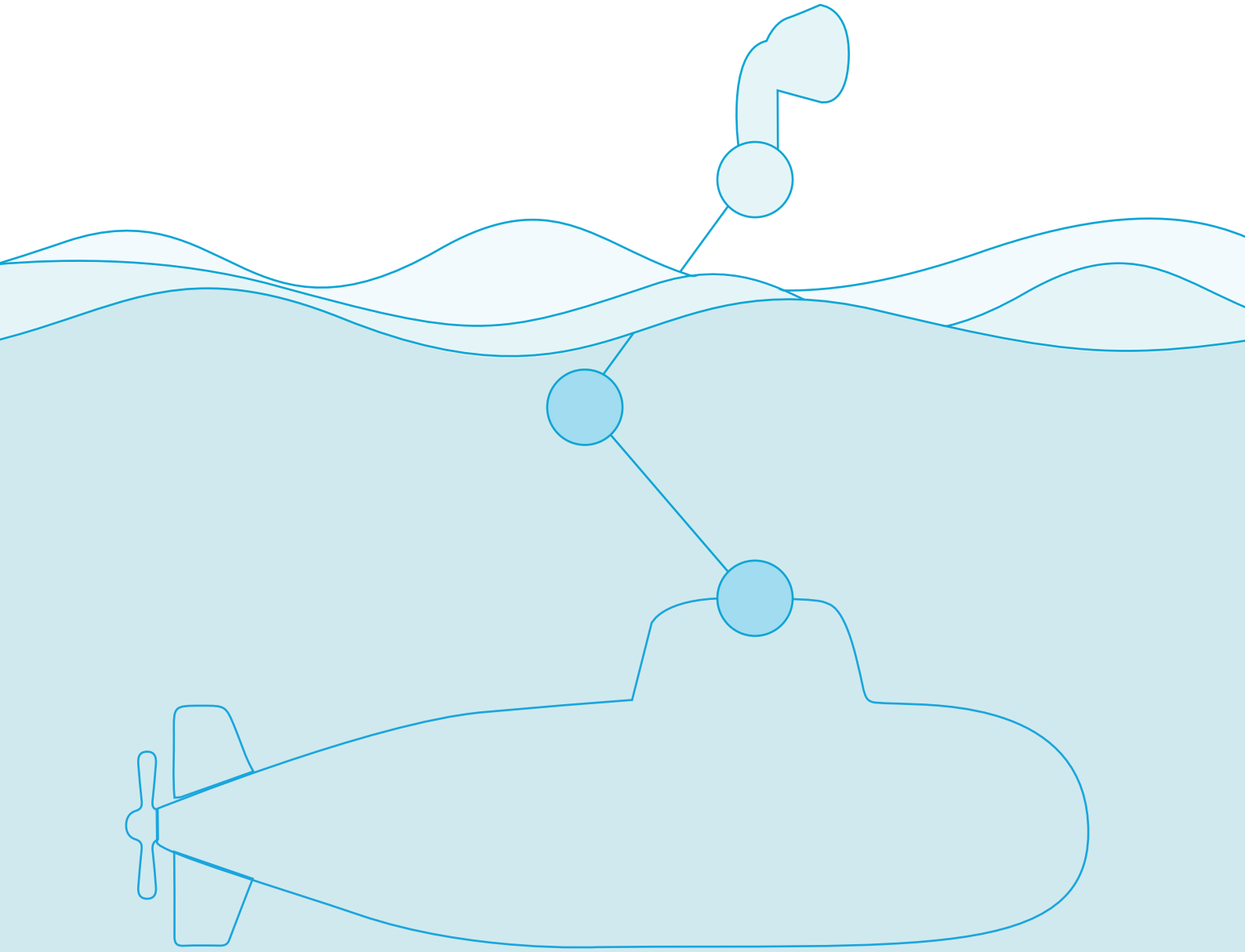
Periscope

Censorship-Resistant Off-Chain Traffic Tunnelling



Emiel de Smidt

Periscope: Censorship-Resistant Off-Chain Traffic Tunnelling

Master's Thesis in Cyber Security

Emiel de Smidt
Supervised by: Stefanie Roos

Distributed Systems Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology, Delft, The Netherlands

Author
Emiel de Smidt

Title
Periscope: Censorship-Resistant Off-Chain Traffic Tunnelling

Supervisor
Dr. S. Roos

Graduation Date
30th of August 2021

Graduation Committee
Dr. S. Roos, Delft University of Technology
Prof.dr.ir. D.H.J. Epema, Delft University of Technology
Dr.ir. S.E. Verwer, Delft University of Technology

Cover Design
Kirsten van der Ham

Abstract

There is an everlasting arms race between censoring bodies and those in its grip. When the censor is employing increasingly sophisticated techniques to digitally monitor and restrict those in its scope, equally sophisticated means to circumvent the digital repression come forward. Those suffering under digital censorship are using nifty ways to escape the censor's grip. Digitally restrictive regimes occasionally still allow access to blockchain applications such as cryptocurrencies, albeit in limited form. Blockchains often enjoy a global nature, but traditional and well established cryptocurrencies such as Bitcoin often do not have high performance. Second layer solutions, also known as off-chain solutions, offer a network of payment channels where transactions can be completed in peer-to-peer fashion with little interaction with the slow blockchain. In this thesis we investigate how Bitcoin's off-chain solution, the Lightning Network, can be employed to circumvent digital censorship.

We introduce Periscope, a protocol that allows for tunneling of internet traffic between two hosts over a stream of micro-transactions embedded with data. Next, we thoroughly analyse its security and the network's suitability from a theoretical perspective. Following this, an empirical evaluation study is done to get an understanding of the performance of the protocol under various circumstances. We hope that Periscope serves as an additional means to access the free internet to those in need.

Acknowledgement

The report in front of you marks the conclusion of my time as a student. It is the result of nine months of effort during a pandemic that has disrupted the student life that I had grown so fond of. Whilst this research has been conducted almost exclusively from the comfort of my studio apartment, it has been far from a solo effort. First and foremost I'd like to express my gratitude to my supervisor Stefanie. Every Friday afternoon, 15:00 sharp, we had great discussions on Lightning and all that we found interesting. Stefanie's advise and help has been of great value to this thesis. Completing a thesis during a pandemic would've been so much more challenging if it weren't for my dear friends Jasmijn, Pravesh, Maikel, Rowdy, and Lex who inspired me to study Computer Science in the first place. The unconditional support that I have received from my family is the reason why studying has always been a joy to me, and for that I am thankful.

Emiel de Smidt
Delft, August 2021

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

In settings where government repression and censorship form a large obstacle to the sharing of information, means of free communications can be limited. Over the last decade, online freedom and rights have slowly deteriorated worldwide [56]. Especially during the global COVID-19 pandemic, where a large share of activities have transformed to digital alternatives, digital surveillance and restrictions have become more prominent [56]. Digital censorship has prompted a large amount of tools and techniques that attempt to tunnel or obfuscate internet traffic to circumvent restrictions. Ideally, a solution is resilient against censors, whilst offering ease of use and a high degree of anonymity. However, current solutions often fail to meet all criteria and those suffering under censorship do not always have many options at their disposal.

1.1 Leveraging Cryptocurrencies

An accessible technology that digitally restricted regions nevertheless often share with the lesser repressed regions are cryptocurrencies such as Bitcoin and Ethereum, albeit in limited form [57]. Whilst cryptocurrencies primarily aim to fulfil financial needs, their underlying technology allows for applications with different purposes.

Compared to on-chain transactions, off-chain transactions offer significant benefits at little compromise [8]. For example, the Bitcoin network suffers from a very low transaction throughput, averaging around at most seven transactions per second over the entire network. The related off-chain solution, the Lightning Network, offers instant (micro)transactions, without reoccurring fees imposed by the network. These features can form the basis of a communication channel, where text and files could potentially be shared between peers. If data can be exchanged in the form of transactions, it is desirable to make it indistinguishable to normal transaction patterns and flows. As information shared on off-chain channels does not need to be broadcasted over the main network, as well as a lack of central entities overseeing the communications [8], such channels seem promising candidates for censorship circumvention.

1.2 Research Question

This research aims to study the potential that off-chain channels such as the Lightning Network have to circumvent digital censorship, and by doing so providing those suffering under digital censorship with another path to digital freedom. To address this, the following research question has been drafted:

"How can off-chain Blockchain protocols be leveraged to circumvent digital censorship?"

To arrive at an answer to this question the following research objectives have been set:

1. Study the impact of digital censorship and how it is enforced from a technical point of view.
2. Study off-chain solutions and select a solution that has potential to carry data and circumvent censorship.
3. Define a set of goals that should be met for an effective censorship-resistant solution that works on off-chain solutions.
4. Design and implement a protocol that meets the set goals.
5. Analyse the protocol from a theoretical perspective to show to what extend the protocol offers security.
6. Empirically study the implemented protocol to evaluate its performance and suitability to tunnel internet traffic.

1.3 Periscope

We introduce Periscope, a protocol atop Lightning that can tunnel network traffic between two hosts over a stream of microtransactions. We define two nodes, a *Periscope* node P as well as a *Submarine* node S. Here, S is the node that wants to tunnel its traffic and use P to proxy the traffic to less restricted regions of the internet. The nodes do not require a direct channel between each other, but instead can use the existing topology of the network to escape the scope of a censoring body. On both ends of the tunnel, internet traffic and service messages are split up in small parts that allow to be embedded in transactions as a custom record. This is possible as the Lightning standard now allows to attach small arbitrary data to transactions that could normally for example be used for the sake of bookkeeping.

From a high level, the tunnelling is realised in the following manner; node S runs the client protocol on the host and configures it as a proxy. At launch, a session request is embedded in a transaction

and sent to a desired Periscope node P and a handshake is performed. Once completed, the client on node S listens for new connections, and informs P of a new connection to be set up to the desired host H. Once P has established a connection to H, traffic is tunnelled over microtransactions from S to H and vice versa with P acting as a proxy. On node S no further interaction is required except from configuring the Submarine clients as a proxy in a browser, and selecting a Periscope node. The Periscope protocol offers a mechanism that allows for tuning the resulting Lightning traffic to a suitable model in order to obfuscate its presence to an overseeing adversary. An existing project, *WhatSat*, leverages this feature to build a chat client [6]. Whereas Whatsat serves as a proof of concept of attaching data for communication purposes, Periscope goes beyond that and demonstrates how an off-chain solution such as Bitcoin's Lightning Network can effectively be used to tunnel internet traffic in a censorship-resistant way.

1.4 Report Outline

The work in this report is structured in the following manner. We start by presenting the reader with the background information on Payment Channel Networks in section 2. Here, the mechanisms of the Lightning Network are discussed, and several aspects that make it a suitable candidate for censorship resistance are explained. After this, a brief overview of censorship and its methods are discussed in section 3. In section 4 related work is discussed that leverage blockchain technologies to evade censorship, or use the Lightning Network to carry data as well. We define a set of requirements and goals for a protocol that leverages off-chain solutions to tunnel internet traffic in section 5.1. We then continue to introduce Periscope and its workings in section 6. After this introduction, an analysis from a theoretical approach is done on the security and privacy claims of Periscope in section 7. Following this, an empirical study is done where the protocol is benchmarked under different conditions to analyse its usability in section 9. Here we also evaluate Periscope's ability to adjust its traffic to a custom model. The report concludes with points of discussion in section 10 and a concluding section with recommendations for future work in section 11.

2 BACKGROUND ON PAYMENT CHANNEL NETWORKS

This section provides the reader with the required background information for understanding the work presented in this thesis related to Payment Channel Networks. We start with providing a brief overview of the blockchain as used by Bitcoin and continue with its secondary network, the Lightning Network, that aims to tackle Bitcoin's scalability issues.

2.1 The Blockchain

Since the publication of the original Bitcoin white paper by the mysterious author Nakamoto in 2008 [49], the topic of blockchains has sparked great interest in both the academic as well as economic communities. The blockchain as introduced by Nakamoto offers security guarantees to a public and decentralised ledger where there is no central authority at play. However, since its inception it has long surpassed its goal and its scale has imposed a significant penalty on the usability of the system. In essence, the Bitcoin blockchain is a community-curated ledger of transactions, where one's current balance can be inferred by tallying all the incoming and outgoing transactions. The Bitcoin network is supported by miners. Miners are nodes who gather transactions and compose blocks of transactions, which are to be appended to a chain of blocks; the Blockchain.

2.1.1 Security, Privacy, and Censorship

The blockchain is maintained by miners all around the world, and the validity of the network relies on whether or not the majority of the miners are honest. The consensus algorithm used by Bitcoin, *Proof of Work*, makes it infeasible for malicious mining nodes to incorporate fraudulent transactions in the blockchain as long as the majority of the computational power is in hands of honest nodes [49].

In traditional banking, a certain level of privacy is achieved due to the transactions not being disclosed to the entire public. However in Bitcoin, the flow of transactions of the entire network is stored in the blockchain and can be considered public knowledge. The privacy that Bitcoin has to offer directly comes from the absence of the final link; the link between the public address and the owner of the address. However, several threats pose a risk to this marginal level of privacy [30].

Multiple nations have taken measures against Bitcoin and other cryptocurrencies. For example, China has banned crypto-exchanges from operating, but that did not make the possession ille-

gal [57, 23]. Regulatory actions against cryptocurrencies are often backed up with claims that cryptocurrencies pose a large financial risks, that the mining imposes stress on the electrical infrastructure, or that it can be used to fund terrorists and other illegal activities [41]. Besides local regulation, there is little that a censor can do to disrupt the blockchain, assuming that no censor will have the means to acquire enough computing power to outperform all the honest nodes combined or distress the network in other ways. However, a censor could directly monitor the network and detect and consequently block Bitcoin-related traffic within their scope.

2.1.2 Scalability and Issues

The underlying consensus algorithm of Bitcoin, Proof of Work, does not scale well. Due to the set amount of transactions that can be encapsulated in a block, as well as the dynamically managed limit of one block being mined roughly every ten minutes, there is a throughput of roughly seven transactions per second worldwide. Furthermore, a transaction is considered validated by the network only after five consequent blocks, resulting in significant latency. No matter how much the network grows, the current consensus algorithm will not allow for a higher throughput or faster confirmation times. Furthermore, to incentive mining nodes to include a transaction in the to-be-mined block, a fee should be included when making a transaction. Hence, the network can become congested and costly quite quickly. If two nodes, Alice and Bob, were to frequently buy something from each other with Bitcoin as the currency, a lot of additional costs and inconveniences are to be taken into account.

It is evident that Bitcoin is very limited in terms of throughput, and the high fees makes it less than ideal for using it as a means to transfer funds on a regular basis. In order to achieve a more usable payment network, a layer-two (also known as *off-chain*) protocol has been proposed called Lightning [53].

2.2 Off-chain Solutions

One of the most well-established off-chain solutions is the Lighting Network [53], Bitcoin's payment channel network that aims to provide faster and cheaper payments.

Bitcoin its underlying consensus algorithm, Proof of Work, makes it inherently unsuitable for an efficient payment network of large scale. The Lightning Network allows participants to exchange currency without constant involvement of a potentially slow blockchain, and instead only interact with the blockchain when opening or closing a payment channel or to settle a dispute.

Two parties that wish to make frequent transactions to each other can construct a direct payment channel, and eliminate the direct involvement of other parties and the Blockchain. This in turn greatly decreases the latency and reduces the transaction costs. The Lightning Network is a

mesh network composed of individual payment channels between parties. The set of all payments channels forms the payment channel network, where nodes can send funds to other nodes with whom they do not share a direct payment channel by propagating a transaction over a route of payment channels.

The Lightning Network is composed of multiple Lightning clients. Most prominent are lnd[1] by Lightning labs, eclair[2] by ACINQ and c-lightning[3] by Blockstream. In an attempt at standardisation, the BOLT specification, *Basis of Lightning Technology*, has been drafted [2]. Here, a description of protocols and a set of guidelines is provided, such that different clients can interact with each other and form an operational network. Differences between clients can be found primarily in parts such as the path finding algorithms [40]. The work presented in this paper focuses on the lnd implementation as it is closely follows the BOLT specification, and is the most prominent implementation [47].

2.2.1 Payment Channels

A payment channel is a connection between two nodes that allows for direct payments between the parties. Two users of the Lightning network can set up a channel by creating a multi-signature commitment transaction that locks funds on the blockchain. These funds can only be released when both signatures are present, i.e. both parties in a channel agree on doing so. This locking of funds establishes a channel. We denote the capacity as the amount of funds deposited on a channel.

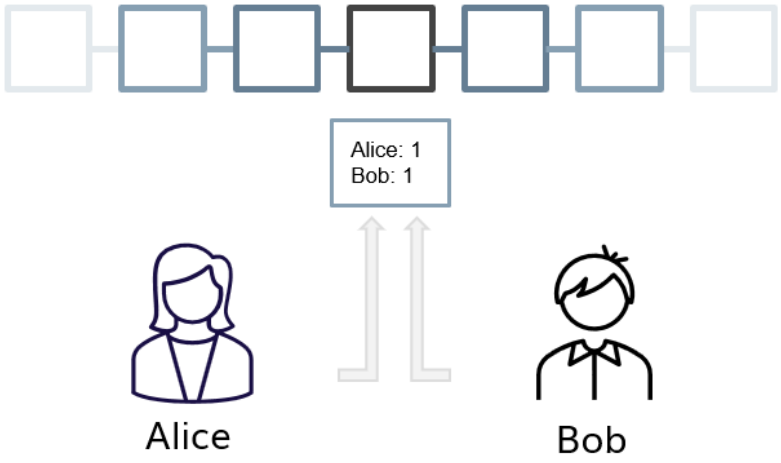


Figure 1: Alice and Bob open a channel by funding a multi-signature wallet on the blockchain.

Both parties of a channel have a balance, indicating the amount of funds that a node could send to the other party. In the Lightning Network the funds are often expressed in Satoshis, the smallest possible division of a Bitcoin (one hundred millionth of a single Bitcoin). A transaction between

[1]https://lighting.engineering/
[2]https://acinq.co/
[3]https://blockstream.com/lightning

the two parties of a channel occurs when they cooperatively agree upon a new state of the mutli-signature wallet its output transactions. If a transaction is made, the payee's balance increases while simultaneously the payer's balance decreases. During its lifetime, this capacity remains the same, whereas the balance shifts over time as the participant make mutual transactions. Once both parties agree, a channel can be discontinued. Now, the final balance is broadcasted to the blockchain which results in the parties obtaining their respective balance.

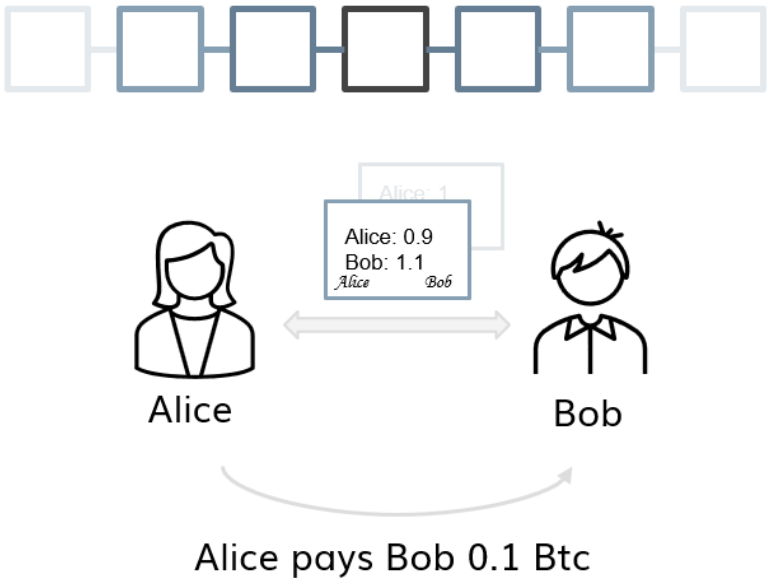


Figure 2: Transaction between Alice and Bob, changing the previous state of the channel.

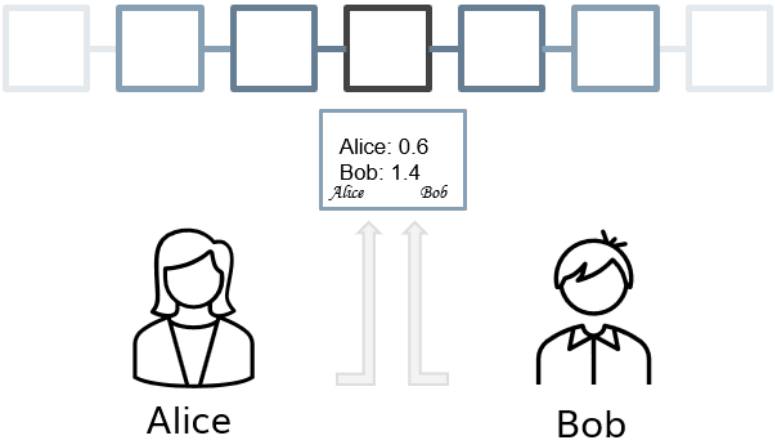


Figure 3: Broadcasting the final standing of the channel, to be payed out as normal Bitcoin transactions.

The Lightning network employs security mechanisms that allows for strict repercussion for fraudulent actions. In the event that a participant of the channel broadcasts an old state that was more favourable, its peer can present a more recent channel state which they have both agreed to with digital signature. Penalties are enforced by the network, which results in the misbehaving peer loosing all of the funds [53].

2.2.2 Multi-Hop Payments

If a payer and payee do not share a payment channel then a transaction can still be realised without creating a dedicated direct channel. Instead, existing channels between intermediate nodes can be utilised to securely forward a transaction. A transaction between the receiving node R and sending node S involves the following phases:

1. R creates and shares an invoice with S that specifies payment details.
2. S calculates a route along suitable intermediate nodes to reach R.
3. Intermediate nodes along the chosen route commit to forwarding the transaction.
4. R reveals a secret value r to its predecessor along the route, claiming the committed funds.
5. The transaction is propagated along the individual connections, from the payee towards the payer.

In a transaction with multiple intermediate nodes involved, the payee node will generate a 32-byte value known as the preimage, r. This preimage r is hashed which yields a payment hash, $h(r)$. This hash value is used to form a conditional payment, where the funds are paid if the original preimage is known. This payment hash, and the amount to be payed are given to the payer S by the payee R as an invoice.

In absence of a direct payment channel, S determines a route towards R over the payment channel network, taking aspects such as costs and reliability into consideration.

After a route has been selected in the previous step, the nodes along the route commit to forwarding the transaction. This commitment heavily relies on the use of Hash time-locked contracts (HTLC). HTLCs restrict the spending of funds until either a specified time has passed, or the preimage of the hash is presented. The sending node constructs a message based on the Sphinx protocol [31], with an extension of per-hop payloads. This packet is constructed in a way that a receiving node can only read the information needed to verify the package, as well as determine the node to forward the package to [10].

What follows is a chain of conditional payments along the route, starting from the payer and ending at the payee. In a nutshell, each node along the route sends its adjacent nodes on the route a transaction that allows for fulfilling the HTLC. The payee node knows the preimage it constructed, and therefore has the ability to claim the funds from their predecessor. By completing this payment the predecessor is granted knowledge of the preimage, and can claim the funds from their respective predecessor along the route in similar fashion. This process continues down the route until the origin payer node is reached.

2.3 Lighting Anonymity and Security

Payment channel networks such as Lightning are built with privacy and security in mind. For a censorship circumvention system, it is desirable if not critical for it to offer anonymity to the user.

Clients on the Lightning Network employ an onion-style routing scheme [10]. Once an intermediate node receives such an onion-message, it can only determine its predecessor as well as successor. Based on the transaction it cannot draw conclusions on who the source or the final destination of the transactions is. This offers a significant privacy benefit that helps in making a censorship resistant communication channel as presented in this work.

The anonymity has been studied in many contexts. It has been shown how certain design choices allow for reasoning and in turn diminished anonymity [37, 44, 45, 58, 40]. For example, the path finding algorithms used by different Lightning clients all aim to be as short and efficient as possible, with little fees imposed by intermediate nodes. Knowing this, a de-anonymization attack can be deployed [40]. The anonymity and privacy offered by the Lightning network has proven to be not as guaranteed as hoped. The work presented in this thesis takes the findings of these reports into consideration, as anonymity is a desirable if not essential property of a censorship-circumventing solution.

2.4 Network Topology and Costs

The Lightning network is a peer-to-peer network that has no centralised critical infrastructure vulnerable to attacks or repercussions that could take down operation of the entire network. The Lightning Network enjoys a large topology, with over 22 thousand nodes and nearly 56 thousand channels at the time of writing[4]. All those nodes and channels could aid the repressed host with finding a suitable route to escape the censor's grip. However, research has shown that the Lightning Network is arguably not a truly distributed network, as a small subset of the nodes hold the majority of stake in the network [42]. This does pose a risk, as a failing or attacked major node could have impact on the network [47]. However, the option often remains to find a route that circumvents major centralised nodes, but this can come at the cost of higher routing fees due to longer routes.

As payments are routed through the network, fees by intermediate nodes are to be payed. An important aspect of the protocol presented in this work is that it is financially accessible, that is, the amount you pay for using the protocol should be limited to reasonable amounts. The Lightning Network allows for making transaction as small as one Satoshi, but intermediate nodes often request a base fee of at least one Satoshi for their forwarding services. Hence, in order to make a successful microtransaction to a node situated elsewhere in the network, one needs to spend

[4] https://1ml.com/

a relatively large share of funds on fees.

The cost of using the Lightning Network are directly related to the current value of the network's underlying protocol; Bitcoin. However, the Bitcoin market fluctuation has little influence on the topological efficiency of the Lightning Network, except for the capacity stored in channels [25].

2.5 Custom Records

The onion-encrypted message send by paying node has hop-specific payloads containing payment and routing details that only the intended recipient can read. Furthermore, the space allocated for these hop-specific payloads is flexible [11]. Due to this flexibility custom records in the *Type-Length-Value* (TLV) can be included in the hop-specific payload, allowing the paying node to send information along with the transaction. Applications can leverage this for various purposes such as bookkeeping. This also opens up the door for experimental features as well such as utilising Lightning transactions for purposes other than financial.

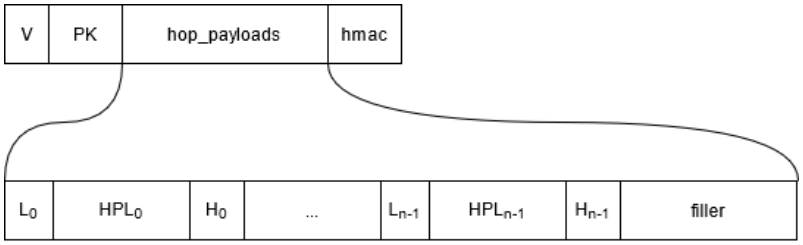


Figure 4: Composition of onion packets and its hop_payloads field.

However, the longer the route between the two counterpart nodes of the protocol becomes, the less data can be embedded. In figure 4 we see that the *hop_payloads* field is composed of multiple variable-length hop-specific payloads with their accompanying fields. As more intermediate hops take part in the route, less room will be available for additional data as the *hop_payloads* field has a predefined length of 1300 bytes. Without additional data, an intermediate *hop_payload* field has a size of l bytes (note, this is not properly specified in the Bolt standards but has been determined to be around 30 bytes), in addition to the single-byte *Length* and the 32 bytes $HMAC$ field. Given that *hop_payloads* field is a fixed size of 1300 bytes, the following formula can be deduced to find out how much data can theoretically be carried depending on the amount of intermediate nodes n:

$$L_{max\_data} = 1300 - n(1 + l + 32) - (3 + 32)$$

Hence, as the required route length grows, more transactions are required to transmit the same amount of data. This results in a higher cost, as more transactions are to be completed along routes where more intermediate nodes ask a fee. However, an increase in hops will make it more challenging for an adversary to trace the transaction. The protocol therefore faces a trade-off between the costs and traceability as well as performance.

2.6 Invoiceless Transactions

The possibility of including custom records as explained in section 2.5 allows for transactions that do not require a prior invoice. Whereas in normal multi-hop transactions the payee creates the preimage and payment hash, an invoiceless transaction requires the payer to construct them. When the payee node receives the onion packet, it can extract the preimage and reveal it to its adjacent node within the chain of commit transactions, setting the shift of funds along the channel in motion. This is aptly named the *KeySend* approach. This allows for spontaneous payments to peers that accept it [15].

3 BACKGROUND ON CENSORSHIP

We present an overview concerning the scope as well as means of internet censorship that users of the Periscope protocol might face. Censoring bodies aim to limit or disrupt the access to or exchange of data. This is achieved by either directly manipulating the information to be censored, or by targeting the means of access to the data. Means to circumvent digital censorship will be discussed in chapter 4 on related work.

3.1 Impact and Relevance

Censoring happens all around the world, for different reasons, and to different extents. Both repressive regimes aiming to control their civilians as well as democratic countries limiting access to content that is deemed harmful censor their inhabitants. The motivation behind censoring often varies. Whereas some censoring regimes restrict access to illegal marketplaces, other might hinder access to sources that contain politically sensitive topics. It can be argued that censoring can be done with good intentions and for good purposes, for example access to content that directly cause harm to individuals or the society. However, such arguments can be misused. For example, this rhetoric is used by various governments to block access to Wikipedia in its entirety [4], where the public is consequently restricted from a large source of information.

3.2 Censorship Scope and Capabilities

Before we define the methods that can be used by a censor, it is important to know what the capabilities and facilities of a censor are. A censoring entity can come in many forms, ranging from parents monitoring their children to restrictive regimes monitoring their civilians. For this work we assume that a state censor has access to the entire digital infrastructure of its region, but no direct endpoint monitoring of the devices used by individuals. A state censor can force full cooperation of an internet service provider, allowing for inspection, rerouting, and blocking of traffic. Such forced cooperation is not unusual and can happen to various extends [18]. It should also not be underestimate what measures authorities are willing to take. For example, Russian legislation allows authorities to isolate the nation from the World Wide Web, similar to what Iran has already done in the past [56].

We define the censoring scope as the digital region of which the censor can be considered to have total control and observatory capabilities. Figure 5 illustrates an example of a censoring scope. Hosts dependent on these systems are within the censoring scope. The protocol presented

in this report aims to offer these restricted hosts means to interact with the internet without the interference of the censor. An important assumption that is being made is that the censoring scope is not an hermetically closed system, and in fact a subset of the World Wide Web. This assumption implies that the censored host wishing to circumvent censorship has means to communicate with hosts outside of the censoring scope, albeit under heavy monitoring or indirectly.

A censorship-circumvention solution would be of little use if it can only operate as long as the adversary is not even aware of its existence. When designing such a solution it is important that the censoring bodies are aware of the potential presence and workings of it. Hence for this report we closely follow Kerckhoffs' principle [52], the protocol's effectiveness should not fall apart once the censor knows how it operates.

To conclude, this work assumes that the censoring body has the resources and intention to monitor hosts on an individual level and can both observe and block traffic if it decides to do so. The censoring body however does not fully isolate all hosts from communicating with hosts beyond the censoring scope, nor does it have the advantage of breaking cryptographic protocols used by the censorship circumvention systems.

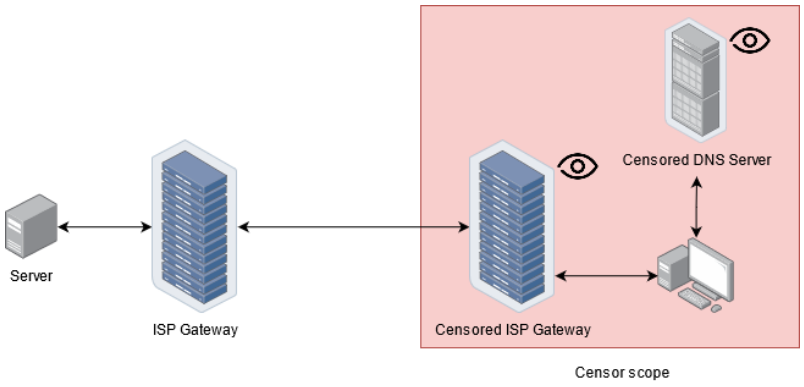


Figure 5: Example of the censoring scope, where the censor has access to critical infrastructure systems such as DNS servers and ISP gateways.

3.3 Methods of Enforcing Censorship

The following subsections briefly cover common censorship techniques. The techniques concern those that rely on filtering, and not on content moderation such as restricting discussions on certain topics on digital platforms. The censorship approaches discussed in this section are relevant as the protocol presented in this report aims to circumvent them.

3.3.1 IP Filtering

A common and relatively straightforward way for a censor to block access to online resources is by maintaining a blacklist of IP addresses. If a censor observes a connection to a blacklisted host, the connection can be interrupted and terminated by sending TCP RST packets, instructing the source host to cease the connection. Significantly more restrictive, but also less common is the usage of a whitelist, where only connections to approved hosts can be made.

3.3.2 DNS Filtering

In order to block interaction with a disallowed host before the interaction is initiated, the DNS requests can be obstructed. Commonly, before a connection is established to a host, its IP address is requested through a DNS request sent to a DNS server. If a censor has control over the DNS server, requests to disallowed domains and hosts can simply be discarded. A censor can restrict access to public DNS servers, and redirect it to their own. DNS filtering creates a strong initial obstruction for users wishing to communicate with a restricted host. Due to the relative ease as well as effectiveness, this approach is quite common and applied by various nations [51].

3.3.3 Deep Packet Inspection

A more sophisticated approach to filter internet traffic is by means of Deep Packet Inspection (DPI), which directly analyses the monitored traffic to draw conclusions on its legitimacy. If a censoring body deems the usage of a service illegal, traffic can be evaluated to arrive at characteristics that can in turn be used to identify usage and take countermeasures. For example, the *Great Firewall of China* inspects traffic and can identify Tor usage based on the ciphersuite presented by the Tor client's ClientHello message [60].

4 RELATED WORK

This chapter provides an overview of work that is related to the work presented in this report. As employing off-chain protocols such as Lightning to carry data is quite novel, published research on using off-chain solutions to communicate is limited. Hence, related work is discussed that touches upon partial goals of this research, or aims to achieve a similar goal in a different way, e.g. by using the blockchain directly.

Firstly we compare the work to more traditional censorship resistant systems, and see that the main distinction between traditional and off-chain systems can be made in terms of being prone to being blocked by the censor. This is followed by an analysis of work that leverages properties of the blockchain such as its global accessibility to provide censorship resistant communication channels, namely *Moneymorph* [46] and *Tithonus* [54].

Following this, work related to the partial goals of the project is covered. We find that Lightning is not immune to attacks nor that it offers complete privacy to the users of it. The techniques that current applications and projects that utilise Lighting for the purpose of transporting data, in this instance chat messages, are compared. Particularly relevant for this thesis are the works related to detectability as well as high volume traffic, as these have posed to be key challenges in designing our protocol. The findings from these studies provide value to the project, as it gives us an understanding of the challenges faced when making a true privacy-preserving and censorship-resistant system.

4.1 Traditional Censorship Circumvention

Whereas blockchain technologies can still be considered quite novel, the practice of circumventing censorship has brought forward many different approaches [39]. This section will lay out a brief comparison between the traditional approaches and the work presented in this report.

A common way to acquire access to web content that is restricted by a censor is by using a VPN, but a censoring body can impose penalties or block critical infrastructure of the VPN provider [19]. Public VPN systems [16, 50], where individuals can host proxy nodes, have come forward. However the traffic of a VPN, either centralised or decentralised, can still be susceptible to fingerprinting, giving room for a censor to block content.

A frequently used system for anonymous web browsing that is also used to circumvent censorship is Tor [32]. With Tor, traffic of the user is routed through multiple proxies and relays, which results in anonymization. The relay nodes that a client can use are publicly available, which allows

a censor to blacklist connections to known Tor-related hosts [20]. In response to this, the Tor Project deploys unlisted entry relay nodes known as bridges that propagate traffic to the rest of the Tor Network. However, censors can actively probe the network to discover Tor nodes, where after a host has been deemed Tor-related, it can be added to the blacklist. Tor can be considered to be in a constant game of cat-and-mouse where the censor blocks critical infrastructure [9]. The work presented in this report relies on the assumption that blocking the Bitcoin network, and in turn the accompanying off-chain solution Lightning, is unlikely to happen due to the related collateral damage [54, 46, 3].

Other ways to hide disallowed traffic and in turn circumvent censorship are obfuscation, mimicry, and appropriation [28]. Obfuscation is the act of making protocol characteristics hard to identify, for example through randomisation or adding redundancy. The assumption is made that a censor is not willing to block traffic of which a censor cannot draw conclusions on the nature. Mimicry takes a different approach and explicitly tries to make traffic indistinguishable from services that are popular and allowed, to avoid being blocked as the collateral damage would be too high. For example, *SkypeMorph* mimics Skype traffic covertly transports data by mimicking Skype calls [48]. Such an approach does however require significant maintenance to maintain indistinguishability as the impersonated service advances. There is also *Protozoa* that covertly tunnels traffic over web streaming services at the respectable rate of 1.4 Mb/s [24]. Whilst *Protozoa* claims to be highly resistant to traffic analysis it does require participation of proxy hosts. A censor can block traffic directed towards such proxy hosts using a straightforward blacklisting technique. The work presented in this report will also require participation of proxy hosts, but as they are reached in a peer to peer manner blacklisting will not be successful if the proxy node is situated outside the censor's scope. An increasingly prominent approach to censorship circumvention is decoy routing, also known as refraction networking [59]. This is a promising line of work that has shown true benefit. However, a critical aspect of such a system is the cooperation of an ISP, which by doing so face the risk of retaliation. Due to the decentralisation of the Lightning Network, the unlikely cooperation of an ISP or other central entity is not required.

The work presented in this paper does not aim to replace the aforementioned systems, but offer an alternative instead. Whereas other approaches could offer higher bandwidth while being easier to use and more cost efficient, the accessibility can not always be guaranteed.

4.2 Layer One Alternatives

Utilising the blockchain to communicate freely or carry data has been covered in literature. Most notably, Minaei et al. [46] have shown how the persistent and global nature of the blockchain from various cryptocurrencies allow for bootstrapping of secure communication channels to circumvent censorship, as presented with the project *MoneyMorph*. *MoneyMorph* is a stegonagraphy bootstrapping scheme that embeds information such as the public key of a censorship circumventing

proxy in publicly available transactions, which can in turn be used to set up a connection. A comparison between different cryptocurrencies is drawn, and it is concluded that the Bitcoin network is less suitable compared to others such as Zcash due to the high fees as well as latency. The recipient is expected to wait for two hours before it can observe the data on the blockchain. Another major downside of this scheme in combination with Bitcoin is that coins are irreversibly lost, also known as *burnt*. This is a common pitfall of schemes aiming to store data on the blockchain [21]. Furthermore, a frequent topic of critique when it comes to storing data on a public blockchain such as Bitcoin is that it unnecessarily increases the size of the blockchain that participants have to download [21]. Minaei et al. discuss that the introduction of off-chain payment channels allows for new means to communicate freely, which is the focal point of this work. By moving towards off-chain payment channels, the fees have become less restricting as well as virtually no latency for a transaction to arrive at the recipient. Furthermore, all the funds spent by the users to communicate will remain within the network as it is used to pay for fees during routing. Aside from transactions related to opening channels that a user of the system could potentially use specifically for this scheme, no exchanged data is permanently stored on the blockchain.

Furthermore, in the work presented by Recabarren and Cubanar [54] the scheme *Tithonus* is proposed that uses Bitcoin gossiping protocols to exchange arbitrary data. Whereas they do not utilise the second layer networks, it also aims to send data as effectively as possible in a censorship resistant way. Tithonus however still has a noteworthy latency, making it more suitable as a back-up option when other solutions such as VPNs are blocked according to the authors. Furthermore, the economic costs of using *Tithonus* is around 9 Satoshis per byte, which is significantly more expensive than the costs of using our system.

Whereas *MoneyMorph* [46] focuses solely on the bootstrapping of a communication channel under the assumption of heavy censorship, *Tithonus* is designed to handle the communication itself. The work presented in this paper shares the goal of achieving censorship aforementioned projects, but shifts its protocol to the second layer payment channels to lessen the financial burden and be less reliant on a potentially slow and congested blockchain. Later in this report the performance will be compared. The work presented in this report serves as a good comparison how utilising the layer two protocol, Lightning, allows for greater throughput compared to layer one solutions.

4.3 Data carrying Off-Chain Transactions

The BOLT standard has only recently specified the inclusion of arbitrary data to a transaction [5, 11]. The standardisation of this allows for efficient data transmission across the Lightning network, independent of which Lightning client is encountered along the route. At the time of writing there are a few projects that utilise Lightning transactions for the primary purpose to send data to a recipient. The notable projects focus on chat messages, and can be considered either a proof of concept or a serious attempt at creating an application that will be used. As using

Lightning for purposes other than financial reasons is relatively new and uncommon, there is yet to form a substantial research body.

Whatsat [6] is a Lightning application that serves as an initial proof of concept for attaching arbitrary data to transactions. Here, two users can chat with each other, where messages are attached as custom records to transactions. This project demonstrates how small text-based messages can be send. In the work presented in this thesis however, key challenges such as high volume, as well the decomposition and ordering of data are taken into consideration, which are not accounted for nor implemented in Whatsat. The way data is included in a transaction is similar to how it is done in this work. As the intention of *Whatsat* is to serve as a proof of concept, no scientific work is related to it.

Juggernaut [7], similar to Whatsat, is a messaging platform utilising Lightning transactions, that incorporates wallet functionality. The project states to be censorship-resistant, however literature backing up this claim is lacking. As the project is publicly available on GitHub, we can see that the way data is attached to transactions seems to be in line with this project as well as *Whatsat* which is expected to be indeed censorship resistant. However, for the Juggernaut project the focus lies on short chat messages rather than high volume internet traffic as the work presented in this report aims to handle.

There are more projects already utilise the Lightning network to send information along with a transaction[22]. However the frequent use case is to encompass information related to a purchase with the transaction. For this work we leverage transactions to carry data, rather than making a payment. The main difference between these projects and the one presented in this thesis is the use case. Whereas Whatsat and Juggernaut focus on sending chat messages, either as a proof of concept or as an attempt at creating a complete messaging platform, this project aims to provide a censorship circumventing protocol to gain access to restricted online resources. To achieve this, different hurdles have to be overcome, such as reliable high volume traffic that respects the ordering of data.

4.4 High Volume Traffic

One of the initial features that the Lightning project aims to offer is to allow microtransactions [53]. This opens up the door for applications that use a large amount of low-value transactions. Example use cases are pay as you use applications, where you continuously send funds that allow for access to a service, e.g. sending a small amount of funding on a short interval to a recipient that in turn grants access to watching a live concert.

Several studies have been dedicated to analysing the efficiency of the Lightning Network, as well as attempting to achieve the highest throughput of transactions. In these studies, the main point of focus is the routing protocol. A recent experiment by Joost Jager, a developer of the Lightning

client LND, attempted to get a feel to what extent the Lightning Network is suitable for high frequency transactions [36]. The conclusion was that the network is not yet capable of delivering high frequency transactions beyond a throughput of roughly thirty transactions per second. Whilst the network might not be capable of delivering a throughput in the order of hundreds per second yet, several areas of improvement have already been identified and the throughput is only expected to increase over time as development continues and not become congested [35]. An improved throughput over the network is beneficial to our protocol, as this would allows for a higher throughput in terms of exchanged data as well.

An important aspect to take into consideration when sending a large amount of transactions in unbalanced quantities between two nodes is the risk of depleting a channel, where all of the funds have shifted towards one side. This is well covered in the literature. While this thesis does not aim to solve the rebalancing problems that lightning faces, it does keep it strongly into consideration.

4.5 Contribution

To conclude, Periscope contributes to demonstrating how a second layer protocol such as Lightning can offer means to access information sources such as the World Wide Web without restrictions imposed by a censor. Where in the original proposition of the Lightning Network [53] as well as in public discussion the potential use cases of micro-transactions are often fantasised about, there are yet to emerge substantial functioning applications. Current studies focus on several relevant aspects of this work, such as security and user anonymity, but leveraging the Lightning Network as a data carrier is limited outside of non-scientific projects. Periscope also demonstrates how to overcome technical hurdles that come with managing data-embedded transactions that are sent at high rates. We thoroughly evaluate the claims made by the BOLT specification [2] and evaluate to what extent Lightning clients and the network are equipped to handle large volumes of transactions. Periscope has the potential to form a world wide community that can aid those suffering under digital censorship in getting access to the free internet.

5 REQUIREMENTS AND SCENE MODEL

Now that an understanding of the required background and related work has been established we formalise the requirements that a protocol should adhere to. Furthermore, we present an adversary model and make assumptions about its capabilities that have been taken into account for both the design in chapter 6 and the analysis in chapter 7.

5.1 Requirements Formalisation

There are many aspects that a censorship circumventing solution has to take into consideration. Certain levels of security and privacy have to be met, whilst aspects such as high performance might be less relevant. We define a set of requirements as well as goals that a censorship-circumvention protocol that uses the Lightning Network should strive for.

Requirement 1. *Transactions carrying data should be indistinguishable from normal transactions.*

In order for data-carrying transaction to be routed over the Lightning Network it is important that intermediate nodes will not treat the protocol's transactions different to normal transactions. Furthermore, indistinguishability of the transactions prevents an adversarious node or overseeing entity from detecting and consequently intervening with the protocol's operation.

Requirement 2. *Transmitted data should have its integrity uphold.*

If an adversary can temper with the transmitted data then there are ways to enforce censorship. It is therefore crucial that this cannot take place.

Goal 1. *Censored users can use the protocol without risking a censor's repercussion.*

Usage of the protocol should be anonymous and pose little risk of being identified by a censor. However, as the Lightning Network is not free of anonymity shortcomings it is unrealistic to achieve a complete anonymous system if we rely on the Lightning Network. We thus should also focus on the detectability of the protocol, minimising the risk of an adversary identifying users of the protocol.

Goal 2. *The protocol should provide users with a reliable tunnel that supports normal browsing activities.*

Reliable communication between the protocol's endpoints is essential for usability. If the protocol does not allow for continuous streams of data than normal browsing activities can not be realised.

Furthermore it is desirable if not essential that the throughput and latency of the protocol allows web content to be loaded in a reasonable time frame. However, as the protocol's main goal is to be censorship resistant, security is deemed more important then performance.

Goal 3. *The protocol should be as cost-efficient as possible.*

The protocol will utilise transactions to carry data between two peers, which requires the sending node to pay fees to any intermediate nodes as well as funds to the destination. Whilst the Lightning Network allows for transactions of very low value, a large stream of transactions might become costly. It is therefore a goal to minimise the amount of transactions required for transmission of data.

5.2 Adversary Model

For the analysis we define an adversary that can take on two distinct positions in its attempts to undermine the protocol, namely in a position to oversee network traffic of nodes participating in the route, or as an intermediate node. The following assumptions are made about the capabilities of the adversary:

Assumption 1. *The adversary can oversee network traffic towards or from nodes located within its scope.*

Censoring bodies are often government agencies who can pressurise internet service providers into providing them with monitoring capabilities. In the analysis of the protocol we grant the adversary with the same capabilities. The adversary can observe all traffic towards or from a node within its scope, but once a node outside the scope has been reached the adversary looses all its capabilities. However, all traffic between hops along a route is encrypted on the transport level, as defined in BOLT 8 [12]. Under the assumption that the lightning clients implement the peer to peer encryption correctly, an overseeing adversary can not read the transmitted onion packets.

Assumption 2. *The adversary can take on the role of a strategically positioned node in Lightning node in the network, but will not fulfil the role of a Periscope node.*

We assume that Lightning nodes under the control of the censor are present within the network, and could be part of the route chosen by the Lightning client to communicate between the Submarine and Periscope nodes. The involvement with the network is an attempt to detect or intervene with the usage of the Periscope protocol.

Assumption 3. *The adversary is aware of the workings and the potential usage of the protocol within its scope.*

The protocol is designed with Kerckhoffs' principle [52] in mind; the adversary should have full understanding of the workings of the protocol yet no capabilities to undermine its security. The assumption is made that the protocol is facing an adversary that is aware that users under its surveillance could make use of the protocol to escape its grip.

Assumption 4. *The adversary is computationally bounded, and has no endpoint monitoring capabilities.*

Where the adversary does become limited in terms of its capabilities is that it has no direct control over the hosts situated within its scope. We assume that for an adversary the hosts are essentially black box models, where it is aware of what goes in and out, but not what happens inside. This implies that the secure keys belonging to the Lightning client of choice are shielded from the adversary. Furthermore, the adversary is not capable of breaking cryptographic techniques deemed secure by the community.

5.3 Lightning Traffic Model

It is also important to define a model for normal Lightning Network traffic in order to reason about aspects such as detectability of the protocol. Arriving at a good representative model is challenging as the transactions on the network are not broadcasted publicly and there are large differences between nodes. We can nevertheless reason about what could pass as normal Lightning Network traffic based on known use cases and statistics of the network.

A frequently used analogy for explaining the benefit of the Lightning Network is a regular customer visiting a coffee company that accepts cryptocurrency for payments. Here the customer and the company share a payment channel to avoid the long delays and fees that would come with paying with Bitcoin over the blockchain. Such a node would be responsible for creating at most a few transactions per day.

If a node initiates large bursts of transactions at irregular intervals it would not be in line with what one would consider common Lightning traffic. An overseeing adversary could observe this and conclude that the Lightning Network is leveraged for activities unrelated to payments and consequently intervene.

A use case proposed in the original Lightning paper by Poon and Dryja [53] is the commodification of services. With micropayments, customers can pay-per-use for services. For example, a streaming service where users pay for the service with a constant stream of low-value transaction as long as they are watching content. However, such services are yet to emerge and gain traction.

We define the model used in this work based on the use case of pay-per-use services. If the tunnelling can take place over a stream of micro-transactions at a stable rate of roughly ten

transactions per second we consider it unlikely to draw an adversaries attention. This stream can have a small amount of fluctuation that could be attributed to the Submarine node participating in the network as an intermediate node. This model will be used by the throttle mechanism as explained in section 6.4.4 to mimic normal traffic. As the network is still rapidly evolving, a more suitable model might be needed over time.

6 PERISCOPE: INTERNET CENSORSHIP CIRCUMVENTING OFF-CHAIN CHANNELS

We introduce Periscope, a protocol atop Lightning that can tunnel internet traffic between two hosts over a stream of micro-transactions. The protocol devises itself in two counterparts that combined allow for tunneling of internet traffic. We define two nodes, a *Periscope* node P as well as a *Submarine* node S that use the Periscope protocol. Here, S is the node that wants to tunnel its traffic and use P to proxy the traffic. The nodes do not require a direct payment channel between each other, but instead can use the existing topology of the network to escape the scope of a censoring body if desired. On both ends of the tunnel, internet traffic and service messages are split up in small parts that allow to be embedded in transactions as a custom record. Adversaries such as censoring bodies and malicious nodes along the route cannot draw conclusions on the observed transactions, making the protocol suitable for censorship evasion.

In this chapter we start by providing a high-level overview of the workings of the protocol, followed by an in-depth explanation of the different components and their functionality.

6.1 High-level overview

The general outline of how internet traffic is tunneled between the *Periscope* node P and the *Submarine* node S over the Lightning network is as follows:

1. Node S runs the client protocol on the host, and configures it as a proxy.
2. A session request is sent towards P and a handshake is performed.
3. The Submarine protocol awaits incoming connections on it's proxy module.
4. Upon reception of an incoming connection, P is informed and requested to make a connection to the requested host.
5. Incoming traffic on both ends can now be tunneled towards their counterparts.
6. Step three to five are repeated for new incoming connections.

The protocol is comprised of three disjoint components, namely the *Periscope*, *Submarine*, and the *Session* module. Where the Periscope and Submarine modules are responsible for interaction with sockets, the Session module is used as an interface to the Lightning Network and is responsible for managing connections and data-carrying transaction. In figure 6 the interaction between different

components is illustrated from a high level. Here, node P and node S are connected through n intermediate nodes that are not directly involved with the protocol.

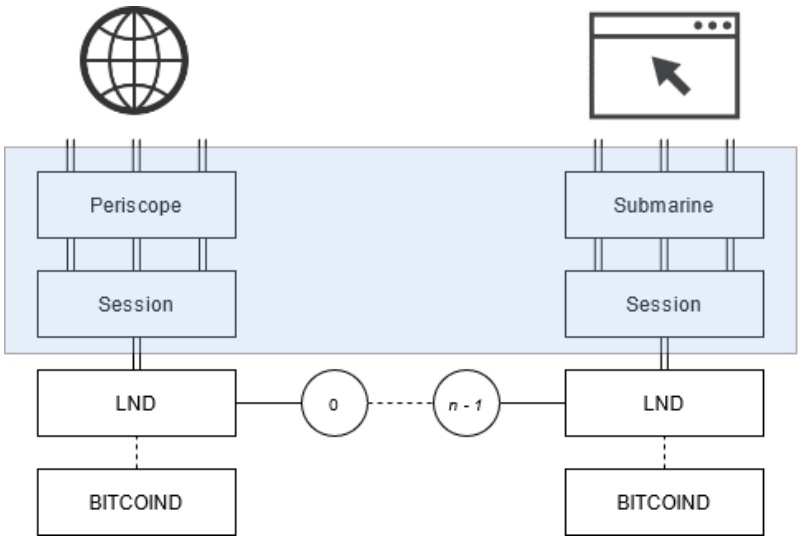


Figure 6: Interaction between different components of the Periscope protocol, blue region marks the Periscope protocol components.

6.2 Submarine Module

A Submarine node, node S, utilises the Submarine module to tunnel it's traffic to the Periscope node P using the Session module. The Submarine component operates on top of the Session module and is responsible for listening to incoming connections, and sending tunnelled traffic to the appropriate socket. During execution, a proxy server is launched that the end user can configure either in the browser or directly on the host itself. With this proxy in place, the Submarine module is informed of any new connections as it receives an HTTP CONNECT message[5] that contains the information required to set up a new connection. With this information a tunnel can be set up in cooperation with the Session module and the Periscope node.

6.3 Periscope Module

A Periscope node utilises the Periscope module to proxy traffic to the appropriate external host and communicate back incoming traffic to the Submarine node using the Session module. Once it is informed by the Session module that a new connection is to be established, a connection is made with the requested external host and from here the traffic can be tunnelled as desired.

[5]https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/CONNECT

6.4 Session Module

Whereas the parts of the protocol residing in the Periscope and Submarine parts mostly concern the management of connections towards the internet or the browser, the session module provides an interface that allows for tunneling the traffic to their counterpart. The session module is responsible for setting up a session between the Periscope and Submarine node, managing of several simultaneous connections over a single stream, and embedding the data and messages required to achieve this into lightning transactions. In the following sections, the aforementioned responsibilities will be individually discussed.

6.4.1 Data Carriage and Messaging

The Lightning standard allows to attach small arbitrary data to transactions that could normally be used for the matters such as bookkeeping as described in section 2.5. Unlike normal Lightning transactions, Periscope focuses on the attached data rather than the transferred value.

The protocol employs two different message types; *Service Messages* and *Data Messages*. Service messages are those that deal with aspects of the protocol such as session setup, connection announcements and connection closings. Data messages on the other hand have the sole purpose of transmitting data belonging to connections.

The embedded network data is encapsulated by multiple layers of encryption, as seen in figure 7. For this work, the protocol opts to exclusively deal with HTTPS traffic as this offers end-to-end encryption. This prevents the Periscope node from having full insight into the transmitted data. Following the HTTPS layer of TLS encryption the data is onion encrypted in line with the BOLT 4 standards [10]. The data is to be embedded as a custom record for the final node's hop-specific payload, which only the recipient has access to. Lastly, following the BOLT 8 standard [12] the traffic between channel endpoints along the route is encrypted on the transport level.

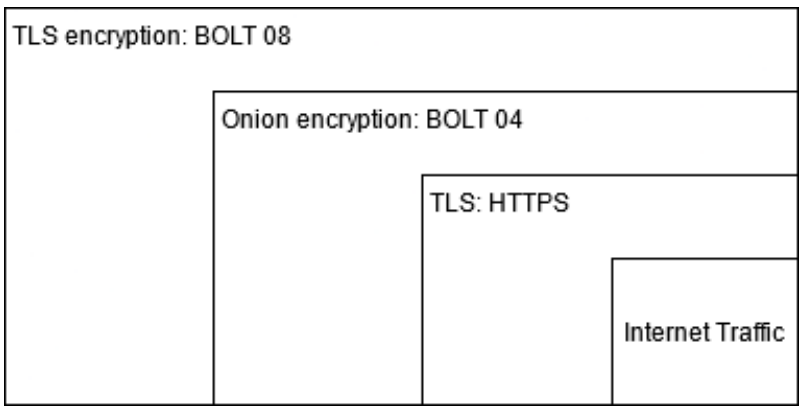


Figure 7: The different encryption layers encapsulating the transmitted network data.

The traditional mechanism of completing transactions with the Lightning Network requires the

payee to create an invoice that is shared and consequently paid by the payer node. This implies that the initiative of a transaction is exclusively for the payee. This mechanism does not lend itself to spontaneous payments, and realising efficient communication channels as a consequence is expected to be troublesome. For example if the periscope node P wants to transmit a packet to the submarine node S, S would have to create and share an invoice beforehand over a potentially insecure channel. To circumvent this significant overhead, the experimental *Keysend* approach is taken, as described in section 2.6. By including the payment hash its pre-image as a custom record in similar fashion to the service and data messages, the submarine can take the initiative to contact the periscope node and send data-embedded transactions without prior arrangement.

6.4.2 Session Management

For the purpose of this research, the establishing of a session has been designed to be a straightforward handshake. The Submarine node presents itself to the Periscope node with its public key. Upon receipt of this request, the Periscope node informs the Submarine node of its decision with either a GRANT or REJECT message. From here, tunneling of traffic can be initiated as described in the following sections.

During a session it is possible that over time an imbalance of funds will occur. This is the case as during a normal web browsing session more content is downloaded than uploaded. The Periscope node will likely send a larger volume of transactions to the Submarine node than other way around. This poses a risk to the channel balances along the route between the two counterparts, as it could result in channel depletion where the balance has shifted towards the Submarine side. A consequence of such a depletion is that no further transactions to the Submarine node can be made. To overcome the risk of such depletion we propose a scheme where the Submarine regularly makes a large transaction to the Periscope node, who in turn can use the received funds for the stream of data-embedded microtransactions. Channel depletion is well-covered in literature [38] and considered beyond the scope of this work. Furthermore, it is up to the Periscope and Submarine nodes to agree upon aspects such as financial compensation for the services.

6.4.3 Connection Management

Setup of a connection: A new connection can be established in the following manner. Once the Submarine module has been informed of a new connection request, the information required to set up a connection on the Periscope node is presented to the Submarine's session module. This includes the external destination, as well as the port number on which the CONNECT message was received. With this information the session module constructs a virtual *Tube* object T_{SP} that has the previously mentioned port number as identifier. Tube objects are used on both ends of the tunnel to have separate message queues for individual connections, for them to be queried by the accompanying socket once the tunnelling has initiated. Following the construction of T_{SP} a service message is sent to the Periscope node that contains the same information. Upon receiving

this service message the Periscope's session module constructs a Tube object as well, T_{PP}. The Periscope's session module now informs the Periscope module about the newly requested connection. Next, the Periscope module establishes a connection to the requested external destination over a newly created socket. Once the connection has been established, the Periscope's session module informs the Submarine that tunneling can be initiated by sending a transaction containing `HTTP/1.1 200 Connection established`. All outgoing data messages are prepended with a tube index (see figure 8), and upon arrival the index is read and the data is directed to the right Tube's message queue. If the socket on the Submarine that initiated the connection is in a readable state, T_{SP} is queried and the `HTTP/1.1 200 Connection established` message is directed to the socket. Once this process has been completed, the tunneling of the connection can proceed. Figure 9 illustrates the aforementioned steps. During a normal browsing session it is common to have multiple simultaneous connections, by using the Tube objects the protocol allows them to tunnel traffic in parallel and still arrive at the appropriate sockets on both ends.

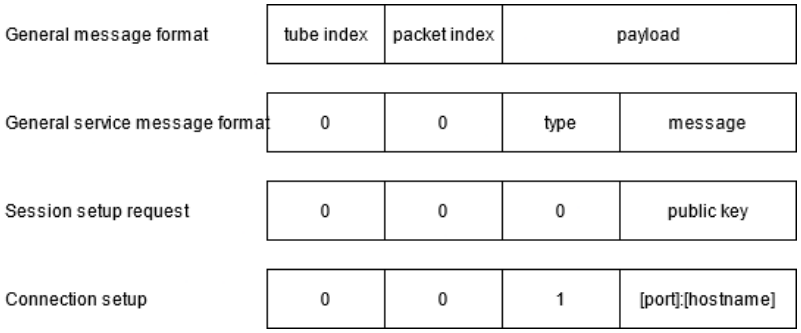


Figure 8: Message formats used by the session module.

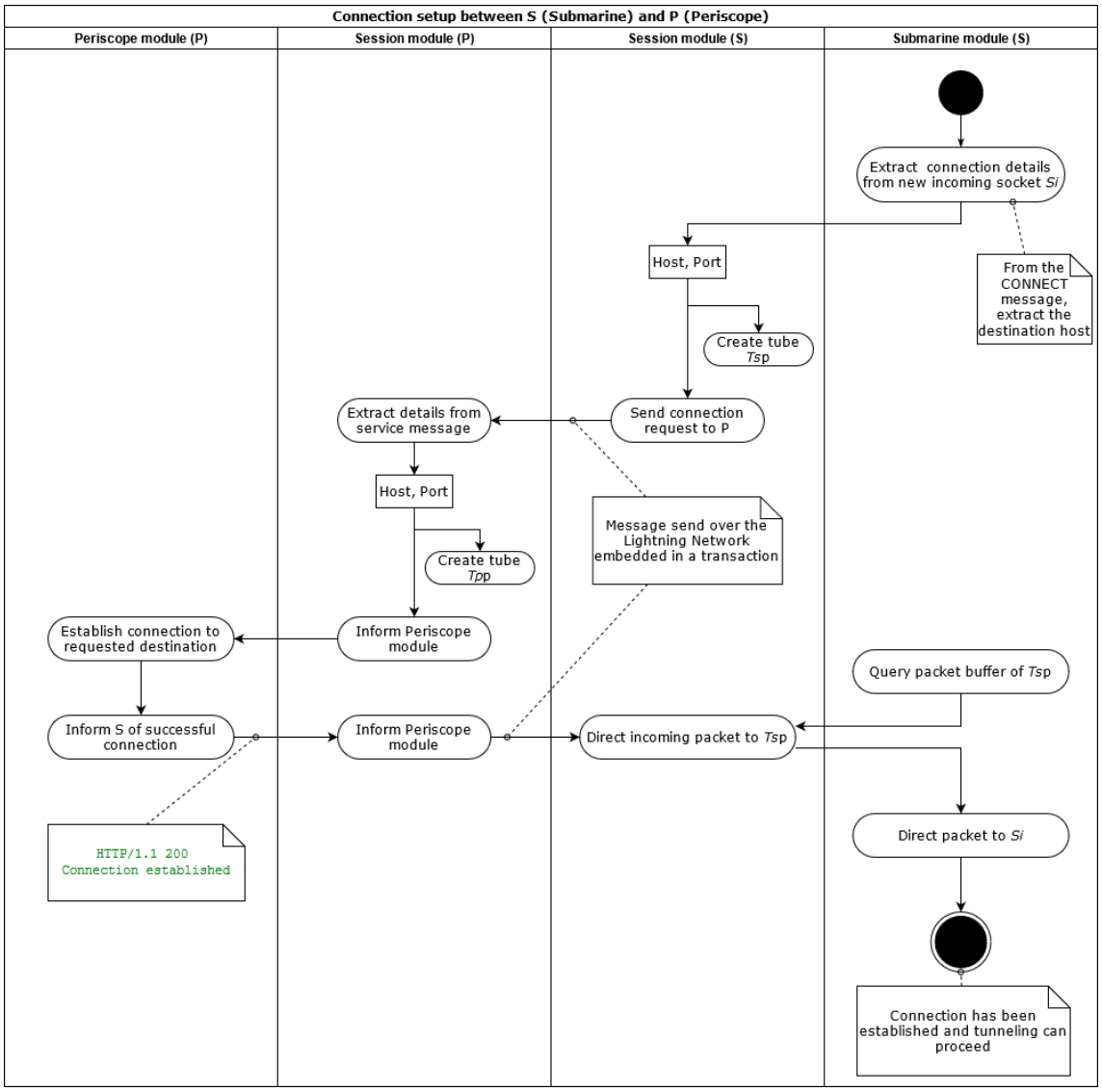


Figure 9: Interaction between different modules during connection setup.

Ordering: A stream of consecutive transactions over lightning are not guaranteed to arrive at the destination in a *"first in, first out"* order. Piping incoming packets to the socket without regard of their original order can result in a corrupted stream and errors in the connection. To overcome this, packet indexes are utilised to keep track of ordering. The Tube object belonging to the specific connection is tasked with both keeping track of a sending index as well as a receiving index. When a socket receives data, the session module consults the appropriate Tube object for the next index, and prepends this to the message to be sent. Upon receiving of a message-embedded transaction, the session layer extracts both the Tube index as well as the message index. The packet is now directed to the appropriate Tube's message queue, where the position on the queue is based on the parsed index. Now, when a socket is in a writable state, the session module presents the appropriate packet if available. Packets that arrive before their preceding packets will not get piped to the socket, and remain in the Tube's buffer until eligible. This process is illustrated in figure 10, where the message formatting can be seen in figure 8.

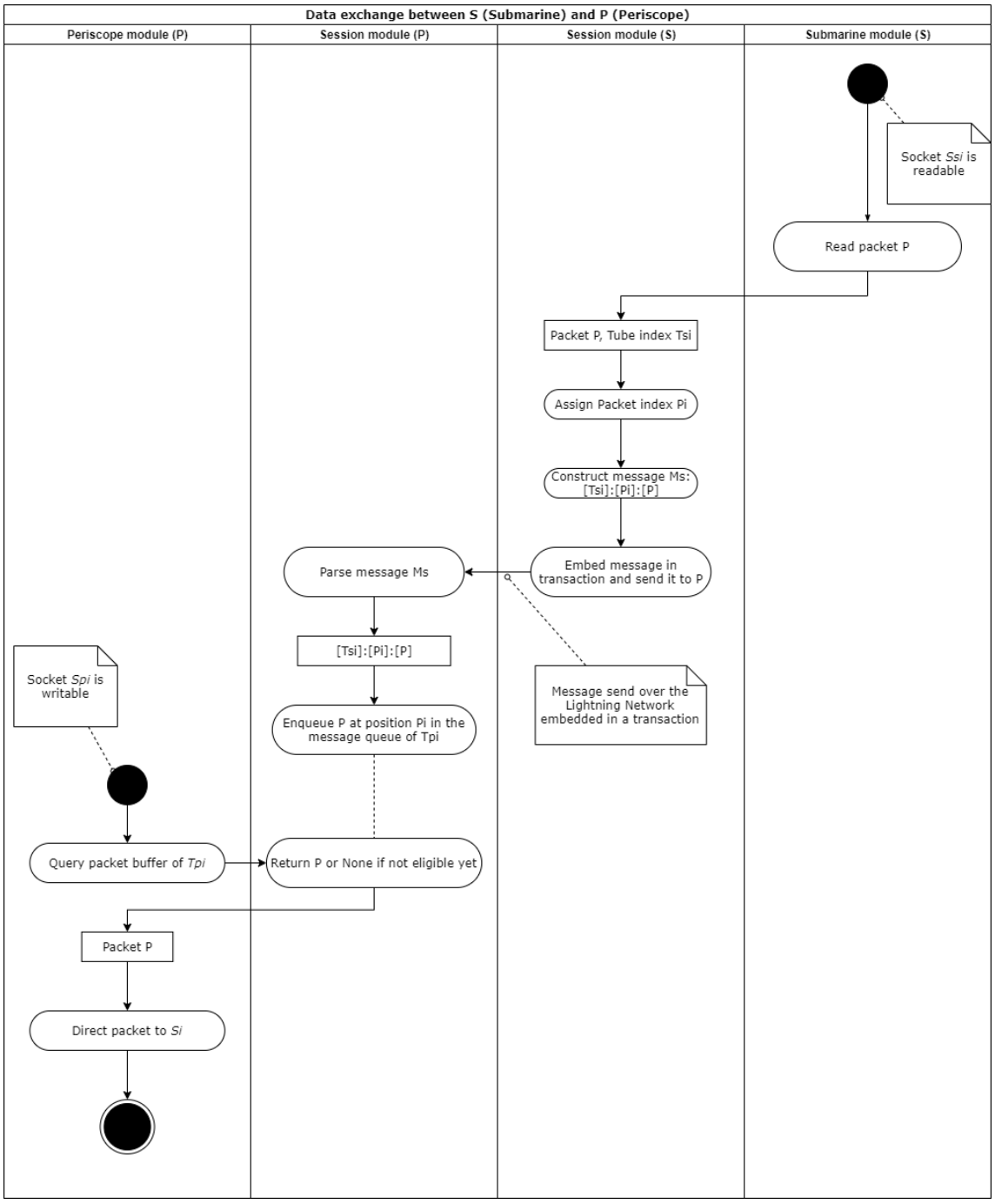


Figure 10: Tunnelling of a packet with respect to order and connections.

Error checking: It is trivial to conclude whether or not the recipient has received a message-embedded transaction. If the (multi-hop) payment would not succeed, error messages will inform us and no funds are deducted from the local balance. This proves to be of great use, as this reduces the significant overhead of constructing an entire stack responsible for checking of faulty transmissions. The Lightning client, and not the protocol, is responsible for making a successful transaction. If a transaction fails another attempt can be made, where the Lightning client is responsible for successfully concluding the transaction. In the unfortunate situation that this can't be realised, the operation cannot proceed as no reliable routes can be found.

6.4.4 Mitigation of Detectability

Whilst the individual transactions aim to be indistinguishable from regular transactions, the network patterns that emerge during usage of the protocol might give an adversary indications that the Lightning Network is used for purposes other than transferring funds. In section 5.3 we have proposed a model of Lightning traffic that the protocol attempts to follow in order to minimise the risk of being detected. The session module is equipped with a configurable throttle mechanism that can be used to adjust the protocol's network traffic to be in line with a predefined model, this aims to achieve goal 1 and make usage less likely to be detected. The throttle takes two variables into consideration:

- **Interval:** The interval at which transactions are send can be limited to a set specification. During normal network activities packets can be exchanged in bursts, by limiting outgoing transactions to a set rate we can dampen such bursts.
- **Filler transactions:** An option to send filler transactions in absence of data-carrying transactions.

In figure 11 the differences have been illustrated. All three diagrams represent the rate of transactions being sent once a website is visited using the Periscope protocol. In the first graph we observe a high spike in transactions that is not in line with normal use cases or the previously defined model from section 5.3. The second graph employs throttling, where the rate at which transactions can be sent is limited to a predefined value. This effectively dampens the intensity of the previously observable peaks. However, the occasional absence of transactions are still notable patterns that can be observed, which are again not in line with normal use cases nor the model. In order to obfuscate these patterns we can opt to send filler transactions. These transactions are sent to keep the rate of transactions at a constant level, even in the absence of normal data-embedded transactions.

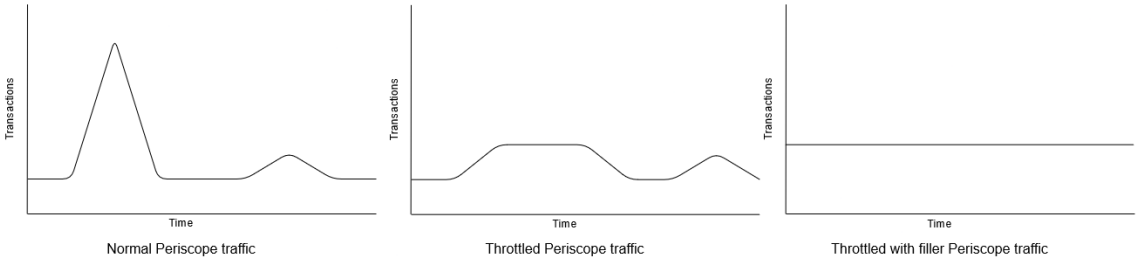


Figure 11: Illustration highlighting the differences between throttling techniques.

The throttling mechanism does come with a trade-off between detectability, performance, and costs. The throttled traffic will take longer whilst requiring the same amount of transactions, favouring goal 1 (detectability) over goal 2 (reliability). If we employ throttling with filler transactions then additional costs will be made and we now favour goal 1 over goal 3 (cost efficiency).

7 ANALYSIS ON SECURITY AND PRIVACY

In the previous section the general outline and workings of Periscope have been discussed. This chapter aims to give a more detailed explanation from a theoretical perspective how the protocol achieves the privacy and security guarantees that are desired for censorship-resistant systems such as Periscope. The analysis relies on the specifications and requirements that Lightning clients should adhere to, as defined by the BOLT standard [13]. We find that these standards offer strong defences against traditional censorship approaches as well as a degree of privacy. However, the Lightning Network and therefore the Periscope protocol faces diminished anonymity when taking side-channel attacks into consideration.

7.1 Security and Privacy

In the following subsections several security properties are discussed, and how Periscope aims to fulfil them. First we argue that data-carrying transactions created by the Periscope protocol are indistinguishable from normal Lightning transactions. Following this, it is shown that the involved parties can be ensured that the transmitted data has not been tampered with. It is also reasoned how the protocol offers a degree of privacy to both counterparts of the protocol, but that full privacy has not been achieved.

7.1.1 Confidentiality of Embedded Data

For a protocol to be considered secure in terms of confidentiality, only those allowed to read or write the transmitted data are capable of doing so. Furthermore, a node along the route that is not involved in the protocol or an ISP overseeing the connections should not be able to draw conclusions on the content of transmitted data or the presence of such. The following analysis is written from the perspective of an adversary that is part of the route chosen by Lightning, and assumes that the adversary does not take attack vectors other than packet analysis such as traffic patterns into consideration. We define the following definition and its accompanying security game:

Definition 1. *A data-carrying transaction is confidential if an adversary cannot do better in the confidentiality game than guess at random.*

Security Game Confidentiality

Game Initialisation

$R : [pk_s, ..., pk_A, ..., pk_r], \beta : \{0, 1\} \leftarrow \text{Setup}(\lambda)$

Oracle query $\Omega(m_1)$

$\tau \leftarrow E_R(m_\beta)$

return $\tau_A \leftarrow D_{\{0,...,(A-1)\}}(\tau)$

Game Finalisation(τ_A)

$\beta' \leftarrow A(\tau_A, \{m_0, m_1\})$

return $\beta = \beta'$

The security game is derived from the *Indistinguishability against Chosen-Plaintext Attack (IND-CPA)* Game [29], where the adversary is allowed to make queries to the encryption oracle. During the setup phase a route R between the sending node s and receiving node r, where the adversary A takes part in, is constructed. Furthermore, β is selected from the set $\{0, 1\}$ at random. The adversary can now presents the encryption oracle Ω a plaintext message m_1 that it wishes to be embedded in a transaction as many times as it desires. The oracle now either embeds the empty message m_0 or m_1 based on the selection of β in a transaction destined for the Periscope along route R. The resulting transaction τ is now onion-decrypted to arrive at the state of the transaction that the adversary would receive when forwarding the transaction: τ_A. Presented with τ_A, the adversary has to determine whether τ_A contains the empty message m_0 or its provided message m_1. The game is won if the embedded message can be identified by the adversary.

In order to reason about this, we first have to understand how and why Lightning allows for embedding data to its transactions. Before sending a transaction along a route R, the sending node s constructs an onion packet that contains the relevant information for each hop along the route, as well as the final receiving node r. The construction of this onion packet happens in iterative fashion. The sending node calculates a shared secret with each hop along the route using Elliptic-curve Diffie-Hellman (ECDH) algorithm on the hop's public key and an ephemeral private key, of which the outcome is hashed using SHA256 to arrive at a shared secret ss_k. With this shared secret, a set of keys are generated (see appendix 12.3).

At the start of the packet construction, 1300 random bytes are generated with the ChaCha20 algorithm [26] as an initialisation packet. From here the onion packet can be iteratively constructed in reverse order, starting with the final destination. For each iteration the sender computes a HMAC H and Length L_i to be accompanied with the hop payload HPL_i. The 1300 packet bytes are right shifted and the three fields are prepended, where the bytes exceeding the 1300 byte limit are discarded. Now, the sender generates a 1300 byte long pseudorandom stream using ChaCha20 in combination with a key derived from the shared secret. The current state of the onion packet is then obfuscated by performing an XOR operation with this stream. This set

of operations is repeated for every hop along the route. As a new ephemeral key is created for each individual transaction, the resulting shared secret will also be different. This will result in a different pseudorandom stream, and therefore no two transactions will be the same even though they share the same payment details.

The variable length hop payload fields contains TLV (*Type-Length-Value*) onion payloads, which is normally used for embedding payment details and routing information. However, the format also allows for flexibility, where clients such as lnd [1] can choose to allow nodes to embed custom records. In figure 12 the composition of an onion packet as defined by BOLT standard is illustrated. For the periscope protocol, data is transmitted as a TLV onion payload and is found in the final hop payload, HPL_{n-1}.

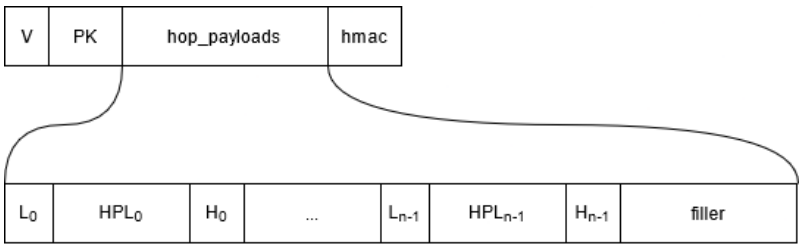


Figure 12: Composition of onion packets and its hop_payloads field.

When an intermediate node i receives the onion packet it can only extract the Length L_i, the hop payload HPL_i and the associated HMAC H_i. This is the case as it can generate the same pseudorandom stream that was used to obfuscate it during the initial construction to apply an XOR operation and retrieve the original de-obfuscated stream. To each intermediate packet along the route the transaction will look like a concatenation of the three extractable fields, followed by obfuscated data that could either be filler data or further information for consecutive hops. After having extracted the relevant details, the payload is left shifted and padded with filling so that the packet remains 1300 bytes. We define the following theorem:

Theorem 1. *Data-carrying transactions created by the periscope protocol are confidential as only the intended recipient is aware of the presence of the data and able to extract it as described in the confidentiality security game.*

The following proof is given with the assumption that all the cryptographic libraries and techniques have been implemented correctly by the Lightning client:

Proof. Upon receiving an onion packet, an intermediate node can only extract the intended information as it can generate the same pseudorandom stream that was used to obfuscate it by the sender using the shared secret ss_k. An adversary node along the route could only conclude that a transaction is carrying data if it can distinguish ChaCha20 from random. ChaCha20 is a variant of Salsa20 that follows the same design principles, but improves at diffusion between rounds [26]. Salsa20 is secure and capable of producing ciphertexts indistinguishable from random [27]. From this follows that an adversary node cannot draw conclusion on the potential presence of a

payload that is not for them as it is impossible to distinguish the remaining packet content from random. Furthermore, the pseudorandom streams used during obfuscation are never used twice, making it impossible for an onion packet to be repeated. The adversary cannot do better in the confidentiality game than guess at random. □

We have shown how Periscope achieves requirement 1 as defined in section 5.1. This makes the periscope protocol resistant against a censor blocking traffic based on the content.

7.1.2 Integrity of Embedded Data

A censorship circumvention solution would be of little use if it cannot guarantee that the data being transmitted has its integrity uphold. An unauthorized node or an ISP having means to tamper with the data directly implies that censorship can be enforced. In section 1 we have shown how the onion packet construction ensures that transmitted data remains confidential. This section however will ignore this property and work under the assumption that an adversary does know which part of the onion packet contains the data transmitted by the protocol. By doing so we are giving the adversary an advantage in its attempt to manipulate the data. We show that the accompanying keyed-hash message authentication codes (HMAC) ensure that a recipient is aware of any malicious actions on the data. With this knowledge, it follows that the protocol provides the users with integrity even before the additional defence mechanisms of the onion packet obfuscation as described in section 1 is applied. We define the following theorem:

Theorem 2. *Data-carrying transactions created by the periscope protocol uphold integrity as any attempt at tampering with the data by an adversary will be known to the recipient and invalidate the transaction.*

As seen in figure 12, the entire packet is accompanied by an HMAC that nodes can use to verify the integrity, as well as the source [33]. During construction of the onion packet by the sender, hop-specific payloads are constructed, as well as an HMAC representative of the state of the onion packet to be forwarded by an intermediate node. The origin node can deterministically know the state of the onion packet that intermediate nodes will forward, as no randomness is involved and the filler data as well as the obfuscation stream are derived from a shared key that both the sender and the intermediate node share. Upon receiving the onion packet, a node can verify the packets integrity by computing the HMAC and comparing it against the received HMAC.

The following proof shows how Periscope achieves requirement 2 as defined in section 5.1. The proof is given with the assumption that all the cryptographic libraries and techniques have been implemented correctly by the Lightning client:

Proof. All onions packets are accompanied with an HMAC, where the key is derived based on the ECDH shared secret between the sender and the receiver. When an intermediate node tampers

with the onion packet to be forwarded in ways other than described by the BOLT 4 standard [10], the pre-computed HMAC will not match and the recipient will be aware of the malicious actions. Under the assumptions that only sender and receiver have knowledge of the shared secret, the receiver can verify that the integrity of the received data is uphold. □

7.1.3 Anonymity

The anonymity that the protocol offers is a direct result of the workings of the used Lightning client. This section will describe the anonymity of the protocol facing differently situated adversaries, a node along the payment route and an overseeing entity such as a censorship-cooperative ISP. Finally we discuss to what extent a Submarine needs to place trust in the honesty of a Periscope node. We first analyse the anonymity with an intermediate node as adversary.

Messages that are sent are constructed using onion encryption, where the Sphinx package format is used. As explained in section 7.1.1 the constructed message offers only the information that intermediate nodes require to successfully complete the transaction, or communicate back errors to the original sender. According to the Bolt 4 specification on the Onion Routing Protocol [10] the onion packets are to be constructed in a way such that the following holds:

Claim: *Intermediate nodes forwarding the message can verify the integrity of the packet and can learn which node they should forward the packet to. They cannot learn which other nodes, besides their predecessor or successor, are part of the packet's route; nor can they learn the length of the route or their position within it.*

While it has been shown that based on the transaction itself this claim does indeed hold, the anonymity diminishes when side channel attacks are taken into consideration. Based on the workings of the routing selection algorithms that different Lightning clients employ, an adversary could reason about the likely source and destination nodes if it is appropriately positioned in the network, with increased certainty if it can collude with multiple involved nodes [40]. Hence, the privacy of lightning transactions, and therefore the usage of the protocol, cannot be fully guaranteed.

We now consider the adversary to be an overseeing entity, such as an censorship-cooperative Internet Service Provider. It would be beneficial for the privacy if the route includes well connected nodes. The Lightning Network has unintentionally become relatively centralised due to emerged central hubs that hold disproportionate amounts of funds [42]. This arguably undesired feature could become beneficial for the operation of Periscope as it would make tracing of transactions more challenging for the overseeing adversary. once a transaction arrives at a well-connected and active node it is processed and obfuscated before it leaves the node again, as part of the onion routing.

An additional measure that can also be taken to diminish the risk of a censor taking actions against a suspected Submarine node the Periscope node also fulfilling the role of an active participant in

the Lightning Network for normal transactions as well. If the adversary has managed to determine that a host within its scope is sending transactions to a known Periscope node, it has to consider the possibility that this is for purposes other than the tunnelling of traffic.

Furthermore, if the protocol is used for censorship evasion it is to be expected that the Periscope node is not situated in the same region as the Submarine node, which will result in the data-embedded transaction stream leaving the censoring scope. If the last intermediate node is situated outside this scope, then the censor cannot conclude with certainty that a transaction stream is destined towards a potentially known Periscope node. However, an overseer could observe that a node within its scope is responsible for generating the Lightning traffic based on the ratio between incoming and outgoing lightning messages.

Now that we have discussed the privacy regarding an intermediate or overseeing adversary we analyse the privacy in the scenario that the Periscope node is either malicious or honest-but-curious and compromised. The possibility of a Periscope node logging the interaction with the Submarine should be taken into consideration. As Lightning transactions are source routed, the Periscope needs to know the Submarine's public key in order to send data-embedded transaction towards it. By design of the Lightning Network one could link a public key to an IP address. With this information the Periscope could log which connections are requested by the Submarine node, and by doing so become in the possession of compromising material on the Submarine node. Hence, there is a degree of trust required by the Submarine node in the Periscope node.

Measures can be taken to counter the exposing of the Submarine's public key to the Periscope node. In the work of *Teechan* [43], a payment channel framework is presented that runs leverages *Trusted Execution Environments* (TEEs) [55] on the host for improved security guarantees. Here, the TEE provides assurance that the ran code and its data remain confidential and have its integrity uphold. A similar route could be taken for Periscope nodes, where any operations with the Submarine's public key remain in such a secure enclave. With a securely created onion transaction the Periscope node would not be able to link the internet traffic to the respective Submarine, and it only knows which node to forward it to. While the implementation of this is considered beyond the scope of this work, it is to be taken into consideration.

We have shown that using the Lightning Network as a carrier of data offers valuable privacy measures against malicious entities, but that a truly privacy-preserving and anonymous protocol has not been realised due to the required trust in the Periscope node as well as side-channel attacks. These weaknesses in anonymity are a problem faced by the Lightning Network, and thus consequently Periscope. The impact of this diminished anonymity is that an adversary could identify Submarine nodes which in turn can face repercussions. Periscope allows users to tunnel the traffic to a host situated outside of its scope. Once outside the scope, a censor can not enforce the commonly used blacklisting methods based on IP addresses or DNS queries. By utilising the Lightning Network, Periscope strives for goal 1 as defined in section 5.1, but it can not be achieved without further measures.

8 IMPLEMENTATION

In the previous section the general outline and workings of Periscope have been proposed. This chapter aims to give a more detailed explanation of how the protocol has been implemented for it to be evaluated. The project has been developed fully in Python and can be found on GitHub where installation instructions are provided[6]. Similar to the composition of the protocol, the implementation is also composed of a separate Periscope and Submarine module that share several helper modules. The project has been developed upon the Lightning client LND [14] as it closely follows the BOLT specification and provides developers with extensive documentation. The following sections will elaborate upon the technical implementation decisions as well as the technologies upon which the protocol relies.

8.1 System Setup

The two actors of the Periscope protocol, the Periscope and Submarine node, communicate over the Lightning Network. In order to interact with the Lightning Network a specific setup has to be realised. The implementation uses the Lightning client LND, which in turn uses Bitcoind to synchronise with the blockchain. In figure 13 the interaction of the Periscope protocol with the Lightning client LND as well as the Bitcoin client Bitcoind is visualised. The following subsections will elaborate on the motivation behind the specific clients used for the setup.

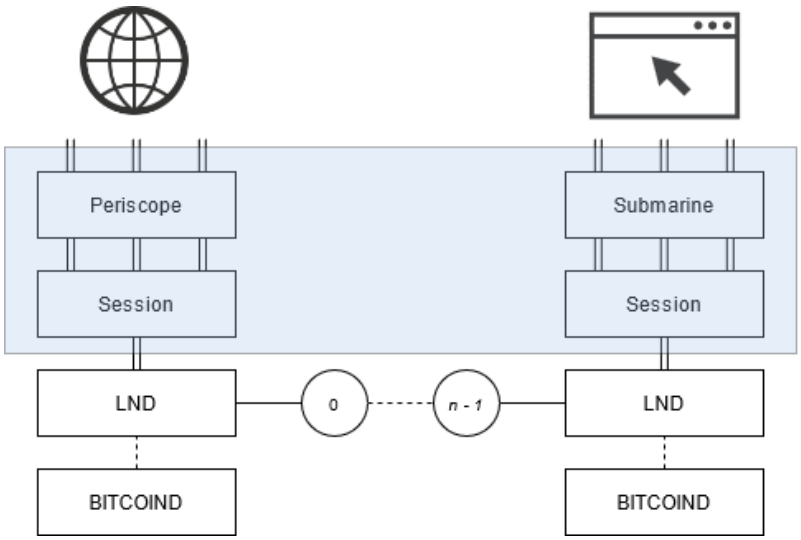


Figure 13: System Setup, blue region marks parts of the periscope protocol, which is situated atop of a stack of lnd and Bitcoind.

[6]URL: https://github.com/emieldesmidt/Lightning-Periscope

8.1.1 Lightning Client

In order to interact with the Lightning Network we have to work with a Lightning client. There are various implementations, of which most comply with the previously mention BOLT standards. The most prominent clients are LND, C-Lightning and Eclair. For this work the LND client has been deemed the most suitable as it is widely used and well documented. LND provides developers with an API which allows for easy interaction with the Lightning Network.

8.1.2 Bitcoin Blockchain

Even though one of the key ideas of off-chain solutions such as the Lightning Network is to have as little interaction with the slow blockchain as possible, some interaction with the blockchain remains necessary for operations such as opening and closing channels or in case of a dispute. For this work we have opted for hosting a Bitcoind[7] node that interacts with Bitcoin's test network. The decision has been made to work with the testnet to mitigate the financial impact of repeatedly opening and closing channels where fees and collateral are involved. Furthermore, it was considered unethical to put unnecessary load on the real network.

8.2 Design

In the previous sections we have discussed the technology surrounding the protocol, this section will cover the technical implementation of notable parts of the protocol implementation in Python.

8.2.1 Session Module

As already explained in chapter 6, both the Submarine as well as the Periscope module make use of a Session module as an interface to the Lightning Network. As the respective Session modules have a lot of functioning in common, we have defined a Session super class [8] where all the shared methods have been implemented. Role-specific session modules have been created that implement relevant functionality and inherit common methods from its super class. This inheritance has been illustrated in figure 14.

[7] https://bitcoin.org/en/full-node
[8] https://docs.python.org/3/library/functions.html#super

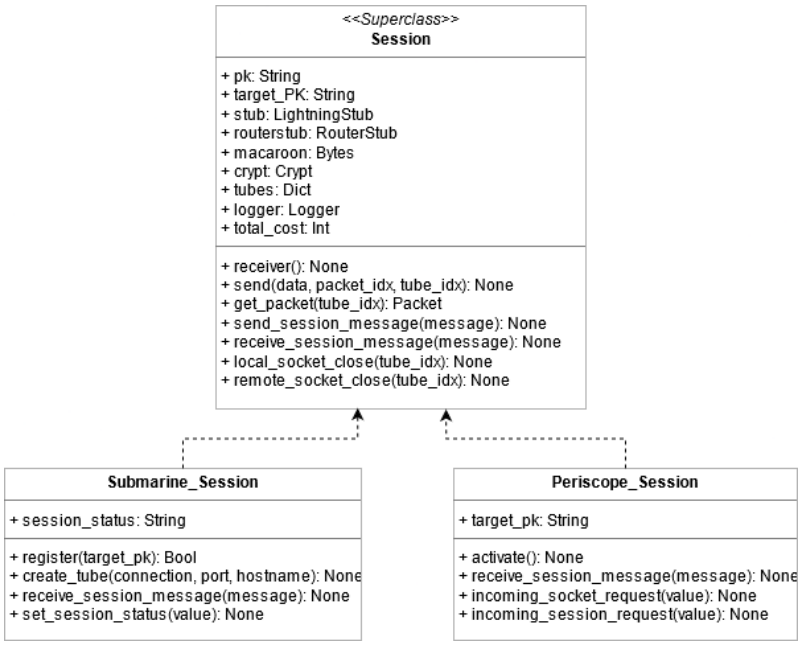


Figure 14: Session interface and the role-specific implementations.

8.2.2 Socket Interaction

Both the Periscope as well as the Submarine node interact with sockets. The socket interaction happens at the level of the Periscope and Submarine module, and not the Session module. Here, a continuous loop is ran that iterates over the active sockets and checks if they are in a readable, writable, or erroneous state.

8.2.3 Data Carriage

The code as found in appendix 12.2 illustrates how data is embedded as a custom record to a transaction, which is send using the *keysend* approach. As explained in section 2.6, a keysend transaction requires the embedding of the payment hash's preimage. The preimage-payment hash pair has been generated using Python's `secrets`[9] and `hashlib`[10] libraries, as seen in listing 1.

Listing 1: Construction of a data-embedded transaction

```
def crypt_pair_generator(self):
    while True:
        preimage = secrets.token_bytes(32)
        phash = hashlib.sha256(preimage).digest()
        yield preimage, phash
```

[9] https://docs.python.org/3/library/secrets.html
[10] https://docs.python.org/3/library/hashlib.html?highlight=hashlib#module-hashlib

8.2.4 High Volume Transactions

The implementation relies on multi-threading to achieve a high transaction rate. The construction of a package, but especially the routing computation and the sending of a transaction are computationally expensive and time consuming. In order to have effective tunnelling one can not wait seconds before sending the next roughly 800 bytes as that would result in close to unusable performance.

8.3 Throttling

Periscope employs an adjustable throttling mechanism as previously explained in section 6.4.4. A throttling mechanism has been implemented in the `throttle` helper object by using a combination of Python's threading library as well as a thread-friendly Queue object. Here, the user can specify at what rate transaction should be made and whether or not blank transactions are to be made when there is an absence of new transactions to be sent. This all aids in the obfuscation of traffic patterns that could be indicative of usage of the protocol. In appendix 12.1 the implementation as used for the evaluation can be found.

9 EVALUATION ON PERFORMANCE

This chapter provides an in-depth evaluation of the protocol under different circumstances. First we investigate to what extent a longer route between the Submarine and the Periscope has on the performance, taking both latency and throughput into consideration. We see that an increase in route length quickly results in degraded performance. Following this we investigate if continuous operation at various transaction rates has an impact on the reliability and performance. Lastly, the throttling mechanism as described in section 6.4.4 is evaluated. Network analysis demonstrate how the throttle can successfully obfuscate network patterns that normally could hint at usage of the protocol.

In this evaluation we take two metrics into consideration when quantifying performance. In order to reason about whether or not the protocol is suitable to fulfil its task of tunnelling internet traffic at a reasonable speed we need an understanding of the throughput that it offers. Another important aspect to take into consideration when investigating the performance is the latency introduced by the protocol.

We consider routes that have at most three intermediate nodes, as we assume that a Periscope node will take effort to be well-connected to central nodes in order to be within short topological distance. Such a route length is furthermore suitable for the vast majority transactions on the Lightning Network [25].

9.1 Evaluation Setup and Approach

For the evaluation we take two distinct setups into consideration; a local testbed as well as the Lightning test network. Evaluating the protocol on the test network would be troublesome due to the many factors that are involved, and it could interfere with the normal operation of the network. The test network is composed of nodes running different hardware, this would make it hard to find routes that are representative of the network as every route will perform different. Furthermore, repeatedly opening and closing of channels is very time consuming. Using a local testbed we have full control over the channels and all nodes have an equal amount of computational resources available.

Hence, for this evaluation we first investigate the performance in a controlled Docker[11] network, and follow with a validation test on the live test network[12]. In studies on the Lightning Network, custom testbeds are often created where the nodes only simulate relevant aspects to the study, such

[11] https://docs.docker.com/engine/install/ubuntu/
[12] https://1ml.com/testnet/

as for example routing. For this evaluation however, all functionality of the Lightning clients has to be taken into account. We therefore resort to using Docker images that have been published by the client developers. The testbed evaluation has been conduct on a laptop equipped with an Intel i7-6700HQ processor with 16 GB RAM memory running Ubuntu Linux 20.04. Here, networks have been simulated by connecting Docker nodes running Lightning clients using Polar[13]. After each ran experiment the network is reset to it's initial state to avoid tests influencing each other. All the intermediate nodes are LND clients, unless stated otherwise. This has been done as approximately 91.3 percent of the network is estimated to be LND [47].

Some test scenarios on the docker testbed differ in terms of computational intensity for the host. As the route between the Periscope and Submarine node grows the load on the host will increase, simply as more containers are active. This would make it challenging to attribute performance differences solely to the working of the protocol. This has been taken into consideration and a testbed has been developed that aims to have a consistent load, allowing for a more fair comparison. With this setup there will be an equal amount of nodes fulfilling the same task at all times. An illustration of the setup for various route lengths can be found in the appendix section 12.5.

9.1.1 Evaluation of Throughput

To evaluate the throughput in a reliable manner a bash script has been written, which can be found in appendix 12.4. The script loads three commonly blocked webpages of various size in repeated fashion using Curl[14], simulating real browsing activities. The webpages are *https://en.wikipedia.org/wiki/Censorship, https://www.bbc.com* and *https://github.com*. Using Curl's output details the download speed is extracted. In order to correctly reason about the throughput, the webpages are loaded ten times each with a pause of a few seconds between them. With the results the average throughput can be determined. Even though the loaded websites are hosted externally, the connection between the Periscope node and the web-server is not expected to be a limiting factor. This is the case as the supported throughput over the Periscope-Submarine tunnel is several orders of magnitude lower compared to that between the Periscope and the web-server (roughly a factor thousand). This will not have a significant influence to the tests results. Furthermore this would be representative of real Periscope usage.

9.1.2 Evaluation of Latency

In order to evaluate Periscope's latency we have included timestamps as a payload to transactions send between the Submarine and Periscope nodes. Upon extracting this timestamp from the transaction it is compared to the current time. The difference between these timestamps make up the latency between embedding and extracting payloads. This way the extra latency that would be introduced by letting the periscope communicate with an external host is circumvented.

[13]URL: https://lightningpolar.com/
[14]https://curl.se/

Furthermore, the exchange of timestamp-embedded transactions is repeated over 2500 times in order to arrive at a representative average, or to evaluate its growth over time.

It should be noted that during usage on the the real Lightning Network, the latency will be higher due to physical distances. The experiments in this section will therefore only give insight into how much the operations of the protocol or intermediate nodes attribute to the latency.

9.2 Impact of Route Length

We start with evaluating how the route length, that is the amount of intermediate nodes, impacts the performance. To reason about the performance both the latency between the Submarine and Periscope nodes as well as throughput are measured as described in section 9.1.1 and 9.1.2. The tests are relevant as the protocol can be expected to be ran under different conditions. For example, a censor might prohibit a Submarine node to interact with a large set of blacklisted Lightning nodes. In such a situation a Submarine node will have to resort to indirect channels of various lengths. It is expected that the latency increases as more intermediate nodes participate in the tunnelling as they all have to forward and complete the transaction. In terms of throughput a decrease is expected for every additional intermediate node as more overhead is introduced.

With the testbed setup we construct a route connecting the Periscope and Submarine nodes, and iteratively increase the route by adding more intermediate nodes. We then proceed to measure the performance metrics of throughput and latency.

9.2.1 Latency

We now investigate the impact of increased route length on the latency between the protocol endpoints. As described in section 9.1.2, a stream of 100 timestamp-embedded transactions is send at the low rate of one per second. This rate ensures that nodes are not handling multiple transactions at the same time, allowing to arrive at a representative average latency

n	Latency (s)	$\Delta n - 1$ (s)	σ
0	0.165	-	0.0055
1	0.299	0.134	0.0057
2	0.431	0.132	0.0087
3	0.564	0.133	0.0092

Table 1: Comparison of average latency for increasing route length with n intermediate nodes (1 TPS).

We also observe an increase in the latency once more intermediate nodes are introduced. It is as

expected that an increase in route length will result in an increase in latency, simply because more Lightning nodes need to interact with the transaction. We see that an additional intermediate node will introduce around 0.13 seconds of latency. Furthermore we see that as the route length increase, the deviation does as well. As more nodes are introduced, the likelihood of having a node introduce latency due to for example a computational disturbance increases as well.

9.2.2 Throughput

We now continue to evaluate the throughput in combination with the aforementioned commonly blocked webpages. As explained in section 2.5, once more intermediate nodes participate in the route there will be less available space for embedding data. Hence we divide the throughput evaluation into two distinct experiments; one with at most 775 bytes (the most that a route with three intermediate nodes can carry), and one where the maximum amount of data depending on the route length. Through experimentation we have found that these are 775, 816, 850, and 891 bytes for 0,1,2, or 3 intermediate nodes. The resulting throughput as well as the average time to load the webpage can be found in table 2:

n	Throughput (kB/s) Max carriage	Throughput (kB/s) 775B carriage	Time (s) Max carriage	Time (s) 775B carriage
0	43.96	38.64	6.87	7.89
1	27.94	25.80	10.95	11.91
2	20.69	19.05	14.68	16.05
3	15.42	15.42	19.70	19.70

Table 2: Comparison of throughput for increasing route length with n intermediate nodes.

We see that the introduction of additional intermediary nodes has a direct impact on the throughput. A path with zero intermediate nodes implies that the Submarine and Periscope nodes share a direct payment channel. As expected we see that such a route offers the best performance in terms of throughput. The moment that the protocol's participants do not share a direct payment channel the throughput diminishes. This decrease in throughput however is not linear due to the fact that as the route increases, the latency between the Submarine and Periscope node increases as well. In equation 1 we see that throughput for a route with n intermediate nodes equals the download size over time, where time equals the latency l_{ps} between the Periscope and the Submarine, the latency l_{ep} between the external host and the periscope, as well as the time required for completing all the transactions. Here, the size of the retrieved content, as well as the time to retrieve, embed, send and extract the transmitted data can be considered constant as they are independent of the route length. As the latency between the Periscope and Submarine increases, as seen in table 1, it becomes more influential for the throughput. In section 9.3 we empirically determine the maximum supported transaction rate and reason about a maximum supported throughput.

$$
\begin{aligned}
Throughput(p, s) &= \frac{size}{time} \\
&= \frac{size}{l_{ep} + l_{ps} + \frac{size}{(TPS \times Carriage)}} \\
&= \frac{size}{l_{ep} + C_{peri}}
\end{aligned} \tag{1}
$$

To conclude, an increase in route length is not desirable for the performance of the protocol. Both in terms of throughput and latency there is a notable impact once more intermediate nodes are included in the route. This degradation in performance can be attributed to the time it takes for intermediate nodes to process transactions. Furthermore, the throughput of the protocol is heavily dependent on the route being chosen by the lightning client. A single node can act as a bottleneck for the entire process. As occasional variances in performance can occur for the intermediate nodes, an increase in route length will increase the risk of having an ill-performing node in the route.

9.3 Impact of Increased Transaction Rate

In this section the protocol implementation will be evaluated under sustained loads of various transaction rates. The experiments will give insight into how both the Lightning clients as well as the protocol are capable of handling different transactions rates. A higher rate allows for an improved throughput, and is therefore desirable. The latency will be measured as described in section 9.1.2, where each run is repeated four times to arrive at a representative average. A continuous stream of 2500 timestamp-embedded transactions is sent from the submarine to the periscope node at various rates.

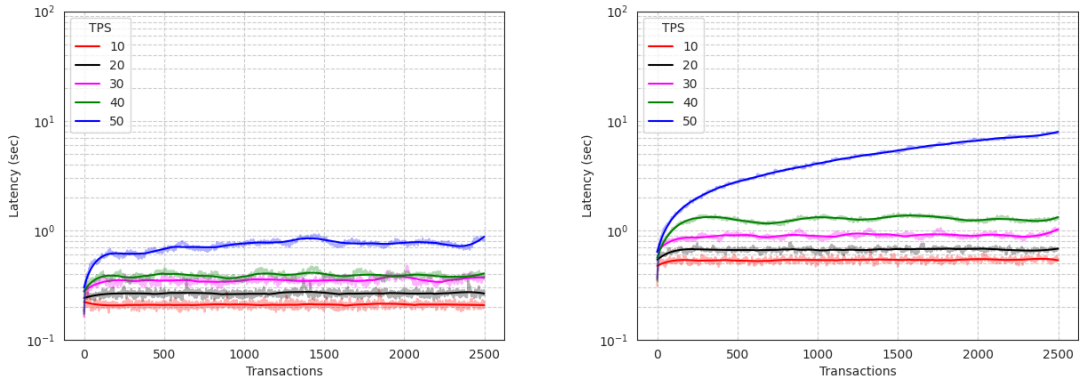
There are multiple factors that could be responsible for a change in latency when faced with a higher transaction rate.

- **The sending node:** The transaction rate can be limited for the sending node, as it might not be computationally capable of sending out the transactions at the desired rate. This can result in a sending-side buffer, introducing extra latency.
- **The route:** Whereas the Submarine and Periscope node might be dedicated to operating the protocol, intermediate nodes are unaware participants. They can be computationally restricted or busy handling other payments.
- **The receiving node:** Once transactions come in faster than it is capable of processing, a buffer will grow increasingly large as more transactions arrive. This will consequently increase the latency between construction of the transaction and the extraction on the receiving side.

We start by analysing the increase in latency once the rate of transaction increases. In figure 15a we show the impact of increasing the transaction rate in absence of intermediary nodes. As

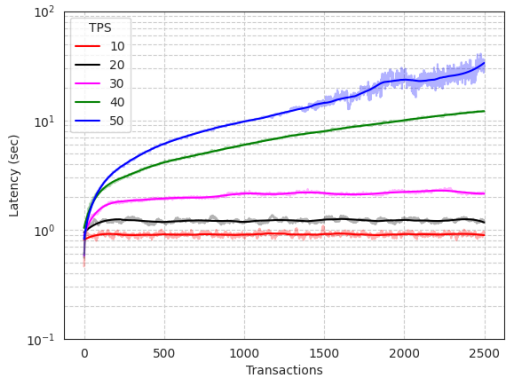
the transaction rate increases at intervals of ten, the latency increases steadily. The Periscope node can send multiple transactions simultaneously by spawning a separate threads. As the rate of transactions increases this becomes computationally more challenging and threads will get less CPU time assigned. As a result the transactions will experience a longer delay.

Once more intermediate nodes are introduced we see that an increase in transaction rate will result in congestion across the route. We see that with intermediate nodes the latency fails to converge at a certain rate as the route can not process the transactions at the rate at which they are sent. As a result the latency between encapsulation and extraction of the timestamps will continue to grow over time as seen in figure 15 b,c, and d. As the route length increases, the minimal transaction rate that causes the latency to continue growing is lower. This is the case as a longer route introduces more channels where congestion can take place.

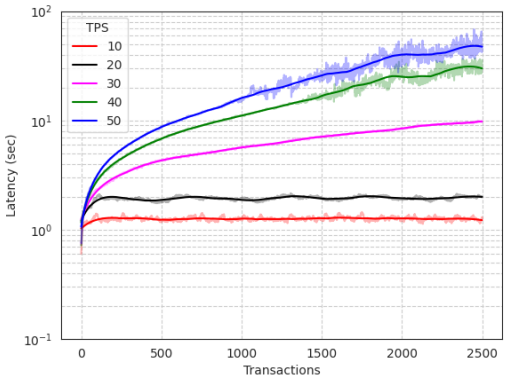


(a) Latency development $n = 0$

(b) Latency development $n = 1$



(c) Latency development $n = 2$



(d) Latency development $n = 3$

Figure 15: Latency graphs for various rates at different route lengths with n intermediate nodes, logarithmic scale.

We now investigate at what rate such an effect can be observed. We now iteratively increase the rate of transactions and investigate whether or not the transaction latency stagnates over time. The resulting maximum supported transaction rates are listed in table 3, where the accompanying

theoretical bandwidth has been calculated by multiplying the maximum carriage by the determined maximum TPS.

n	Maximum TPS	Bandwidth (kB/s)
0	60	53.46
1	42	35.7
2	32	26.11
3	23	17.94

Table 3: Approximation of critical transaction rates for various route lengths with n intermediate nodes.

The results are in line with our expectations; the best performance can be found when the route between the Submarine and Periscope node is as short as possible. The latency will be lower and more consistent due to the less intermediate nodes handling the transactions, and the throughput will be higher. The listed rates might not be representative of performance on the actual Lightning Network due to variances in computational power of the hosts, as well as higher latencies. The experiments do however give a general impression of what the resulting effects are of increasing the transaction rate.

9.4 Client Comparison

We now proceed with an investigation if there is a notable difference in performance between different clients participating as an intermediate node between the Submarine and Periscope nodes. As the Lightning Network is composed of different clients it is very likely that different clients take part in the route between the Submarine and Periscope nodes. Even though all clients adhere to the BOLT standards, the differences in implementations could result in minor differences regarding latency and throughput.

Whilst the Periscope protocol has been configured to run atop the LND Lightning client, we can easily compare different clients acting as an intermediate node. Using the Docker setup we connect a Submarine and a Periscope node with either a LND, C-Lightning or Eclair client as an intermediate node. We evaluate the throughput as well as latency in similar fashion to previous evaluations. The results can be found in table 4.

Client	Throughput (kB/s)	Latency (s)
LND	28.88	0.29
C-Lightning	36.74	0.22
Eclair	6.49	0.24

Table 4: Comparison of throughput and latency for different clients.

The results indeed show differences between the client implementations. Most notable is the

difference in throughput that Eclair offers compared to the others. Hence, having an Eclair node participating in the route will drastically reduce the performance. However, only a small fraction, roughly two percent, of the nodes participating on the Lightning Network are Eclair [47]. This will make the unfortunate situation of having to tunnel through a Eclair node unlikely. In order to determine why a code analysis is required, but that is beyond the scope of this work.

9.5 Detectability

In this section the traffic patterns will be evaluated, with a focus on the influence of the throttling mechanism as described in section 6.4.4. Whilst the transactions themselves give no indication of data being embedded, network patterns might still give away the impression of ongoing tunnelling. As described in section 6.4.4, the throttling mechanism aims to mitigate the likelihood that a censoring body can draw conclusions on the patterns of the traffic between two nodes.

For this evaluation we inspect the network traffic belonging to the Submarine node. In similar fashion to previous experiments we load the three commonly blocked websites four consecutive times each using a Bash script with Curl to simulate browsing activities. Using Wireshark[15] a network capture has been made, and frequency diagrams have been constructed.

The throttling mechanism adjusts the transaction rate to a predefined model. For this evaluation we define a node sending ten transactions per second for a a pay-as-you use service to be the model. We provide a comparison between three different situations:

- **Unrestricted:** No throttling applied, all transaction occur at the maximal rate that the docker containers support.
- **Restricted:** Throttling applied, the frequency of transactions is limited to a rate of 10 transactions per second.
- **Restricted with obfuscation:** Throttling and filler transactions applied, the frequency of transactions is limited to a rate of 10 transactions per second. In case no data is ready to be sent, empty transactions are sent.

In figure 16 a comparison is presented that shows the impact of the throttling mechanism. Figure 16a clearly shows the composition of the simulated browsing behaviour; loading three different websites four consecutive times, with a small break between the websites. Spikes in network traffic coincide with the web requests being done. Such patterns could indicate usage of the protocol to an observer. If we now proceed to apply restrictive throttling, the previously seen spikes are dampened. In figure 16b it can be seen that the spikes are spread out over a longer period of time. This implies that it will take longer to transfer the same amount of data. Even though

[15] https://www.wireshark.org/

the spikes have been made less recognisable, certain indications of network tunnelling can still be seen such as the gaps between loading the webpages. If we now apply dummy transactions to fill potential transaction-voids, we can see that the tunnelling is further obfuscated in figure 16c. In addition to the lower performance, this extra step of obfuscation will also result in higher costs as more transactions need to be sent. In table 5 the trade-off between the different techniques is highlighted.

Technique	Throughput (kB/s)	Transactions	Total duration (s)
No throttling	25.24	4846	255
Throttling	7.10	5061	655
Throttling with filler	6.93	13450	660

Table 5: Costs and time comparison between different throttling techniques.

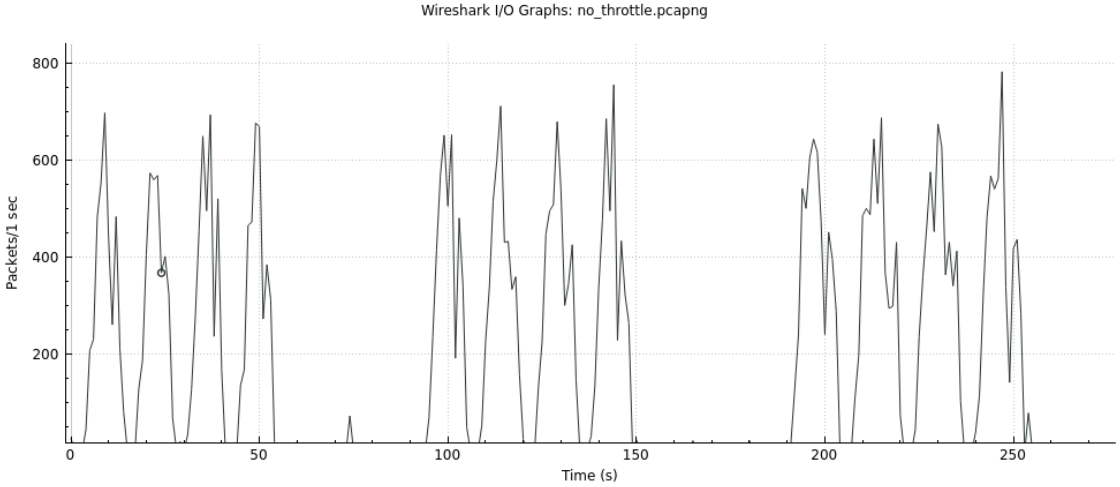
To conclude, the throttling can successfully obfuscates patterns that could be indicative of usage of the Periscope protocol. The resulting Lightning activity is more in line with what we have defined in section 5.3 as expected legitimate Lightning traffic. However, the trade-off comes forward between performance, costs, and detectability.

9.6 Test Network Validation Testing

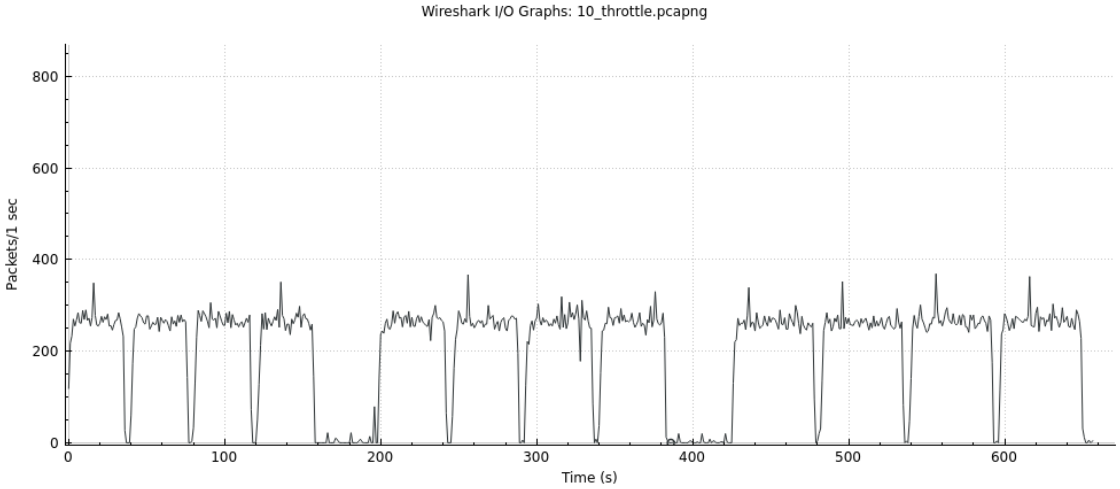
Now that we have evaluated several aspects of the protocol and its implementation on a local testbed we validate its working and performance on Lightning's test network. Two hosts with roughly the same computational power will be functioning as the Submarine and Periscope node, and are either directly connected through a shared payment channel or have an intermediate node between them. We expect an increase in latency as the nodes are not on the same network, but also an increase in throughput. This is expected as the LND clients now enjoy more computational power.

We validate the working of the protocol outside a local testbed by connecting the two hosts through an intermediate well-connected node situated in Australia[16] on Lightning's test network. With the same throughput benchmarking script as used throughout this section we arrive at an average throughput of 14.92 kB/s and a latency of 1.89 seconds between embedding and extracting a transaction's payload. This has demonstrated that the periscope protocol is equipped to tunnel traffic over the test Lightning Network, and therefore also likely the real network.

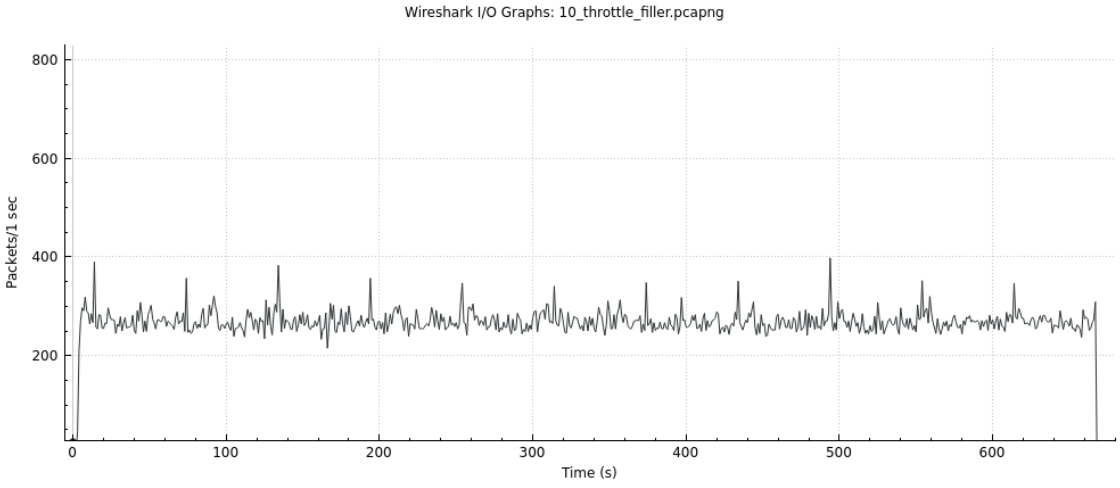
[16]https://1ml.com/testnet/node/038863cf8ab91046230f561cd5b386cbff8309fa02e3f0c3ed161a3aeb64a643b9



(a) No throttling



(b) Throttling



(c) Throttling with dummy transactions

Figure 16: Network graphs highlighting the differences between throttling techniques.

9.7 Comparison to Alternatives

Now that we have a reasonable impression of what performance Periscope is capable of delivering we can compare it against traditional censorship approaches, as well as those that leverage cryptocurrencies. We assume that Periscope does not employ throttling, and that there is one intermediate node involved. We have shown how one intermediate node results in a latency of 300 milliseconds in section 9.2.1, but as this is on a local testbed. For a more fair comparison we assume that all connections along the route introduce an additional 50 milliseconds of latency, and thus work with a latency of 400 milliseconds.

In terms of performance the Periscope protocol does not come close to that of modern and commercial VPN solutions. On average, common VPN providers offer their users an average latency of 200.72 milliseconds and can transfer data at a rate of 13.53 MB/s [34]. For periscope the performance lies significantly lower with 400 ms of latency. In terms of throughput we demonstrated that roughly 35 kB/s can be realised, making Periscope over 380 times less efficient. However, commercial VPN solutions can have critical infrastructure be blocked by censoring authorities, or face legal consequences [19]. Periscope is not susceptible to such threats due to the decentralised nature of the Lightning Network.

A project that shares more similarities with Periscope is the Tor anonymity network. Both Periscope and Tor use layered encryption to pass traffic along its network. The Tor project has a dashboard where performance metrics are published[17]. Depending on which onion server one uses for Tor, a throughput of roughly 1.25 MB/s can be realised when downloading a file . Whilst this throughput is closer to Periscope than it is to a commercial VPN, it is still roughly 35 times as fast as Periscope. The Tor project also publishes round-trip latencies; the time it takes to get an HTTP response to a request. We divide this in half and compare it against the time between embedding and extracting a packet for Periscope. A rough estimation using the provided data gives us a latency of 175 milliseconds in one direction towards a public server. This is again significantly better than Periscope's 400 milliseconds.

If we now compare our throughput to that of protocols that imitate legitimate protocols we see that performance becomes very similar. For example, the *SkypeMorph* protocol has a throughput of 43 KB/s [48]. *Protozoa* covertly tunnels traffic over web streaming services, and manages to do so at the relatively high throughput of 1.4 MB/s [24]. An advantage that Periscope has over imitation protocols is that it does not need to keep up-to-date with the legitimate protocol being impersonated. Furthermore, the peer-to-peer routing of Periscope allows for a higher degree of anonymity.

In comparison with censorship-circumvention solutions that use the Blockchain we see a clear improvement. While no exact throughput numbers are presented in *Tithonus* [54], the authors

[17] https://metrics.torproject.org/torperf.html

state that it does not aim to be a protocol for tunneling traffic as it does not have reasonable latency. *Tithonus*' gossip transactions relay times are in the order of tens of seconds, and no direct endpoint-tunneling scheme is proposed. In terms of costs the benefit of using off-chain solutions becomes even more clear: *Tithonus* requires a costly 1 Satoshi per transmitted byte [54] whereas the Periscope protocol can embed roughly 800 bytes in a single Lightning transaction at the cost of 1 Satoshi (assuming one intermediate node requests this as a fee). It would not make sense to have a communication channel directly on the blockchain, as transaction times will take at least multiple minutes. Instead, it can be used to publicly list information that can be used for further censorship circumvention endeavours. *MoneyMorph* [46] proposes a scheme that embeds bootstrapping information on various blockchains using public-key steganography. In a Bitcoin transaction *MoneyMorph* can embed 20 bytes, which then has to be published to the blockchain. *MoneyMorph* does not focus on how a bootstrapped communication channel will look like, hence we can not make a comparison.

10 DISCUSSION

In this work we have presented Periscope, a protocol for tunnelling internet traffic over a stream of microtransactions in an attempt to circumvent censorship. However, there are concerns and aspects that could hinder its operation that are yet to be discussed. This section will provide an overview of concerns and questions that have come forward during the research. Following this a larger scheme will be briefly envisioned.

10.1 Points of Critique

Below an enumeration of concerns and arguments against the protocol can be found, followed by some elaboration or nuance.

Critique 1. *The cost of tunnelling traffic quickly becomes significant.*

The cost of operating the protocol depends on the fluctuating price of the Bitcoin, as well as the fees asked by intermediate nodes. For example, the Wikipedia page on Bitcoin is roughly 1 MB, this would require roughly 1200 transactions worth one Satoshi, excluding fees. If we are tunneling with one intermediate node requesting one Satoshi as a fee, the total cost of loading this webpage will be at least 2400 Satoshis (roughly 0.94 euro). Depending on the importance of the article and the severeness of the censorship one might be willing to pay this toll, but it shows how other activities such as video streaming will be financially infeasible quite quickly. Unless a Submarine and Periscope share a direct channel, or intermediate nodes are generous in terms of fees, usage of the protocol will indeed be quite expensive. Furthermore, in order to reduce the risk of channel depletion it is important to have outgoing channels that have a large capacity or repeated rebalancing.

Critique 2. *The Lightning Network is not capable of providing help to all in need of censorship resistance.*

Even though there are many nodes taking part in the network, it is not equally distributed around different regions of the world. Most nodes are situated in western countries, which often experience less censorship. The table below illustrates this unbalanced distribution[18].
However, the network is growing and more nodes are spawned on a frequent basis, nevertheless the current state could hinder the effectiveness of the protocol for those situated in sparsely connected regions as geographical distance will introduce extra latency.

[18]Data taken from https://1ml.com/location?type=continent

Continent	Node Count
North America	1,581
Europe	1,663
Asia	161
South America	23
Oceania	40
Africa	20

Table 6: Node distribution over the continents.

Critique 3. *The Lightning technology has not fully matured yet, and the network as well as the clients are not capable of delivering a pleasant browsing experienced.*

In the evaluation we have seen that a throughput of 35 KB/s can be realised with one intermediate node. This throughput is relatively low, but suitable for modest browsing activities. This performance strongly degrades when more intermediate nodes become involved. However, the aim of the project is not to compete with modern VPN solutions but instead offer yet another mean to circumvent censorship. Hence, the option to escape the censoring scope is deemed far more important over having high performance. The impression is nevertheless given that the technology is not at a state that fully supports Periscope's use case. For example, all transactions have to be written to disk. Whilst this does not pose a security risk, it becomes inconvenient during extensive tunneling. Occasional transmission errors can result in a corrupted client state, which requires troublesome recovery and a potential loss of funds. This is not unexpected: clients such as lnd are still in a beta phase during the time of this study (v0.13.1-beta for lnd[19]). As the network and the technology matures further, it will become better suited for Periscope.

Critique 4. *Authorities could take repercussions against the entire Lightning Network or cryptocurrencies due to the undesired imposed censorship circumvention.*

For this report we have explicitly worked under the assumption that an adversary is aware of potential usage of the protocol. In line with the related projects of *Tithonus* and *MoneyMorph* we deem it unlikely that an authority would take the decision to ban cryptocurrencies as a whole due to heavy financial implications.

Critique 5. *The protocol could be used for malicious purposes.*

Periscope, similar to many other censorship resistant solutions, faces the argument that it could be used for malicious purposes. This is a debate with very valid arguments for both sides. Any attempt at preventing evil-intentioned usage is a lost cause where we would ironically find ourselves to be censoring users.

[19]https://github.com/lightningnetwork/lnd/releases

10.2 A Larger Scheme

We have shown how the Lightning Network can be leveraged to tunnel internet traffic between two nodes. This could theoretically pave the way for a larger scheme where there is a broad community of Periscope nodes offering their services to Submarine nodes wishing to tunnel their internet traffic to other regions. As an incentivisation Periscope nodes could ask a fee for their services, which can be easily realised through transactions. This community can use the Lightning Network as its foundation to communicate, and serve as a truly distributed world wide VPN. Hosting such a community would require various supporting applications, such as a place where Submarine nodes can find Periscope nodes that offer tunneling services. Here, the services of the periscope nodes can be rated by the community.

As the main purpose of this work was to investigate whether or not off-chain solutions would support tunneling of internet traffic we leave further details of such a scheme to the imagination of the reader.

11 CONCLUSION

The work presented in this report shows that off-chain solutions such as the Lightning Network are promising candidates for traffic tunneling. We have demonstrated how the Periscope protocol can effectively leverage the Lightning Network to tunnel internet traffic between two peers. Periscope is a novel use case of Lightning applications, where challenges such as ordering and high-frequency transactions have been overcome.

We have thoroughly evaluated whether or not the Lightning Network allows Periscope to achieve secure and anonymous tunneling in section 7. Here we have shown that by design the Lightning Network is robust against many censoring attempts, but that the anonymity challenges facing the Lightning Network also have implications to Periscope. Whilst the set requirements in section 5.1 on indistinguishably as well as integrity have been met, the goal of guaranteed security against a censor's detection and repercussions has not been fully met.

In terms of performance it does not meet modern standards such as commercial VPN solutions or that what the TOR browser can offer, but it has proven to be capable for light web browsing activities. The protocol meets the goal of reliability best under specific conditions; as the topological distance between the Submarine and Periscope node increases then the performance in terms of throughput and latency decreases. Utilising transactions for carrying data has a direct cost associated to it, and rebalancing of channels along the route might be needed over time. Cost-wise Periscope is nevertheless significantly cheaper than protocols that write data directly on the blockchain, while also being much faster.

Periscope allows those that are digitally-censored to escape the censor's grip and enjoy the World Wide Web without restrictions. Whilst there are many existing solutions that have the same goal of helping digitally restricted people escape their censor's grip, Periscope is to our knowing the first to leverage off-chain solutions such as the Lightning Network. The protocol does not aim to replace the countless censorship-circumvention solutions, but serve as yet another alternative. A larger scheme has been theorised where a community of Periscope nodes offer their tunneling services to those in need.

11.1 Future Work Recommendations

The work presented in this report serves as a good introduction to exploring the potential of off-chain traffic tunneling. However, several aspects have not been investigated that could further solidify the answer to the research question. Furthermore, interesting questions related to the Lightning Network in general have come forward. Suggestions for further research have been listed below:

Recommendation 1. *Evaluate the performance on the real Lightning network over the test network.*

For both ethical as financial reasons the experiments have been performed on the test network. It is to be seen to what extend the test network is representative of the true network in terms of topology, willingness to open channels, as well as performance.

Recommendation 2. *Investigate the potential of Atomic Multi-path payment channels.*

Recently Atomic Multi-path (AMP) payments have been included in the BOLT standards [10]. Here, a transaction can be completed simultaneously in parts over multiple distinct routes. Whilst this is beyond the scope of the research, it does look promising for an increased bandwidth but will likely come at the costs of higher fees as more nodes become involved.

Recommendation 3. *A financial study on the costs of streaming microtransactions.*

As the value of transactions decreases, the greater the relative cost of network fees becomes. That is, when one wishes to transmit one Satoshi from node A to D, A will have to pay a total of three Satoshis (assuming a fee of one Satoshi per intermediate hop). The question comes forward to what extend the Network is suitable for high-frequency streams of microtransactions.

Recommendation 4. *Investigate the potential of alternative off-chain solutions.*

For this research we have explicitly investigated the potential that the Lightning Network has to tunnel traffic. However, several other major cryptocurrencies have their own off-chain counterpart. Similar studies on alternatives such as Ethereum's Raiden [17] could provide an interesting comparison.

Recommendation 5. *Investigate if the throughput differences between the clients poses a risk to Lightning's privacy.*

In section 4 we have discovered that there is a significant difference in throughput between the different Lightning clients. This could be used as an attack vector in attacks on Lightning's privacy.

REFERENCES

[1] Announcing lnd v0.9.0-beta! — Lightning Labs. URL: https://lightning.engineering/posts/2020-01-21-lnd-v0.9/.

[2] BOLT standards. URL: https://github.com/lightningnetwork/lightning-rfc.

[3] Can China Contain Bitcoin? — MIT Technology Review. URL: https://www.technologyreview.com/2017/12/11/146816/can-china-contain-bitcoin/.

[4] Censorship of Wikipedia - Wikipedia. URL: https://en.wikipedia.org/wiki/Censorship_of_Wikipedia.

[5] invoices: expose custom tlv records from the payload by joostjager · Pull Request #3742 · lightningnetwork/lnd. URL: https://github.com/lightningnetwork/lnd/pull/3742.

[6] joostjager/whatsat: End-to-end encrypted, onion-routed, censorship-resistant, peer-to-peer instant messaging over Lightning. URL: https://github.com/joostjager/whatsat.

[7] Juggernaut. URL: https://www.getjuggernaut.com/.

[8] Layer 2 — Lightning Network — MIT Digital Currency Initiative. URL: https://dci.mit.edu/lightning-network.

[9] Learning more about the GFW's active probing system — Tor Blog. URL: https://blog.torproject.org/learning-more-about-gfws-active-probing-system.

[10] lightning-rfc/04-onion-routing.md at master · lightningnetwork/lightning-rfc. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md.

[11] lightning-rfc/04-onion-routing.md at master · lightningnetwork/lightning-rfc. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md#packet-structure.

[12] lightning-rfc/08-transport.md at master · lightningnetwork/lightning-rfc. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/08-transport.md#encrypting-and-sending-messages.

[13] lightningnetwork/lightning-rfc: Lightning Network Specifications. URL: https://github.com/lightningnetwork/lightning-rfc.

[14] lightningnetwork/lnd: Lightning Network Daemon. URL: https://github.com/lightningnetwork/lnd.

[15] lncli+invoices: experimental key send mode by joostjager · Pull Request #3795 · lightningnetwork/lnd. URL: https://github.com/lightningnetwork/lnd/pull/3795.

[16] Psiphon — Uncensored Internet access for Windows and Mobile. URL: https://www.psiphon3.com/en/index.html.

[17] Raiden Network. URL: https://raiden.network/.

[18] Russia: Growing Internet Isolation, Control, Censorship — Human Rights Watch. URL: https://www.hrw.org/news/2020/06/18/russia-growing-internet-isolation-control-censorship.

[19] Russia will block VPN providers who don't comply with a blacklist request — MIT Technology Review. URL: https://www.technologyreview.com/2019/06/10/135019/russia-will-start-blocking-major-vpn-providers-from-next-month/.

[20] Tor partially blocked in China — Tor Blog. URL: https://blog.torproject.org/tor-partially-blocked-china.

[21] View of Data Insertion in Bitcoin's Blockchain. URL: http://www.ledgerjournal.org/ojs/ledger/article/view/101/93.

[22] Y'alls: Lightning Network powered publishing. URL: https://yalls.org/.

[23] From the legal nature of Bitcoin to discuss the arbitration thinking of Bitcoin dispute, 7 2020. URL: https://www.bjac.org.cn/news/view?id=3769.

[24] Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communica-tions by Parasitizing on WebRTC. 2020. doi:10.1145/3372297.3417874.

[25] Ferenc Béres, István A Seres, and András A Benczúr. A Cryptoeconomic Traffic Analysis of Bitcoin's Lightning Network. Technical report. URL: https://1ml.com.

[26] Daniel J Bernstein. ChaCha, a variant of Salsa20.

[27] Daniel J Bernstein. Salsa20 security.

[28] Cecylia Bocovich and N C Sa. Recipes for Resistance: A Censorship Circumvention Cookbook. Technical report.

[29] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. LNCS 5479 - Advances in Cryptology - EUROCRYPT 2009. 2009. doi:10.1007/978-3-642-01001-9{_}35.

[30] Mauro Conti, Kumar E. Sandeep, Chhagan Lal, and Sushmita Ruj. A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys and Tutorials*, 20(4):3416–3452, 10 2018. doi:10.1109/COMST.2018.2842460.

[31] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *Proceedings - IEEE Symposium on Security and Privacy*, pages 269–282, 2009. doi:10.1109/SP.2009.15.

[32] Roger Dingledine. Tor: The Second-Generation Onion Router. Technical report.

[33] Carlos M Gutierrez and James M Turner. FIPS PUB 198-1 The Keyed-Hash Message Authentication Code (HMAC). 2008.

[34] J. Han and D. Wren. VPN Products Performance Benchmark, 3 2021. URL: https://www.passmark.com/reports/VPN_Products_Performance_Benchmarks_2021_Ed1.pdf.

[35] Joost Jager. Lightning Node Performance: Exploring the Path to 1000 TPS, 4 2021. URL: https://bottlepay.com/blog/bitcoin-lightning-benchmarking-performance/.

[36] Joost Jager. Lightning Node Performance: Testing TPS, 3 2021. URL: https://bottlepay.com/blog/bitcoin-lightning-node-performance/.

[37] George Kappos, Haaroon Yousaf, Ania Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. An Empirical Analysis of Privacy in the Lightning Network. 3 2020. URL: https://arxiv.org/abs/2003.12470v3.

[38] Rami Khalil and Arthur Gervais. Revive: Rebalancing Off-Blockchain Payment Networks. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017. URL: https://doi.org/10.1145/3133956.3134033, doi:10.1145/3133956.

[39] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M. Swanson, Steven J. Murdoch, and Ian Goldberg. SoK: Making Sense of Censorship Resistance Systems. *Proceedings on Privacy Enhancing Technologies*, 2016(4):37–61, 10 2016. URL: http://content.sciendo.com/view/journals/popets/2016/4/article-p37.xml, doi:10.1515/popets-2016-0028.

[40] Satwik Prabhu Kumble, Dick Epema, and Stefanie Roos. How Lightning's Routing Diminishes its Anonymity. 7 2021. URL: https://arxiv.org/abs/2107.10070v1, doi:10.1145/3465481.3465761.

[41] Global Legal Research Directorate staff and Law Library of Congress. Regulation of Cryptocurrency Around the World. Technical report, 2018. URL: http://www.law.gov.

[42] Jian-Hong Lin, Kevin Primicerio, Tiziano Squartini, Christian Decker, and Claudio J Tessone. Lightning network: a second path towards centralisation of the Bitcoin economy. *New J. Phys*, 22:83022, 2020. doi:10.1088/1367-2630/aba062.

[43] Joshua Lind, Ittay Eyal, Peter Pietzuch, and Emin Gün Sirer. Teechan: Payment Channels Using Trusted Execution Environments.

[44] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, T U Wien, and Srivatsan Ravi. Concurrency and Privacy with Payment-Channel Networks. Technical report.

[45] Stefano Martinazzi and Andrea Flori. The evolving topology of the Lightning Network: Centralization, efficiency, robustness, synchronization, and anonymity. *PLoS ONE*, 15(1), 1 2020. doi:10.1371/journal.pone.0225966.

[46] Mohsen Minaei, Pedro Moreno-Sanchez, and Aniket Kate. MoneyMorph: Censorship Resistant Rendezvous using Permissionless Cryptocurrencies. *Proceedings on Privacy Enhancing Technologies*, 2020(3):404–424, 8 2020. URL: https://content.sciendo.com/view/journals/popets/2020/3/article-p404.xml, doi:10.2478/popets-2020-0058.

[47] Ayelet Mizrahi and Aviv Zohar. Congestion Attacks in Payment Channel Networks. 2 2020. URL: https://arxiv.org/abs/2002.06564v4.

[48] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: Protocol obfuscation for Tor bridges. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 97–108, 2012. doi:10.1145/2382196.2382210.

[49] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. Technical report. URL: www.bitcoin.org.

[50] Daiyuu Nobori and Yasushi Shinjo. VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls. *Proceedings of the 11th USENIX Symposium on Networked Systems Design and Implementation (NSDI '14)*, 2014.

[51] Paul Pearce, Uc Berkeley, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, Vern Paxson, Paul Pearce Ben Jones, and Frank Li Roya Ensafi. *Global Measurement of DNS Manipulation*. URL: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/pearce.

[52] Fabien A. P. Petitcolas. Kerckhoffs' Principle. *Encyclopedia of Cryptography and Security*, pages 675–675, 2011. URL: https://link.springer.com/referenceworkentry/10.1007/978-1-4419-5906-5_487.

[53] Joseph Poon and Thaddeus Dryja. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. Technical report, 2016.

[54] Ruben Recabarren and Bogdan Carbunar. Tithonus: A bitcoin based censorship resilient system, 9 2018. URL: https://content.sciendo.com/view/journals/popets/2019/1/article-p68.xml, doi:10.2478/popets-2019-0005.

[55] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted Execution Environment: What It is, and What It is Not. 2015. URL: https://hal.archives-ouvertes.fr/hal-01246364, doi:10.1109/Trustcom.2015.357i.

[56] Adrian Shabaz and Funk Allie. The Pandemic's Digital Shadow — Freedom House. URL: https://freedomhouse.org/report/freedom-net/2020/pandemics-digital-shadow.

[57] Samuel Shen and Andrew Galbraith. China's furtive bitcoin trade heats up again, worrying regulators, 3 2021. URL: https://www.reuters.com/article/us-crypto-currency-china-idUSKCN2AT201.

[58] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network. Technical report. URL: https://eprint.iacr.org/2020/303.pdf.

[59] Benjamin Vandersloot, Sergey Frolov, Jack Wampler, Chuen Tan, Irv Simpson, Michalis Kallitsis, J Alex Halderman, Nikita Borisov, and Eric Wustrow. Proceedings on Privacy Enhancing Technologies ; 2020 (4):321-335 Running Refraction Networking for Real. Technical report.

[60] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is Blocking Tor. Technical report. URL: `http://www.cs.kau.se/philwint/static/gfc/`.

12 APPENDIX

12.1 Throttle Helper Object

```
import threading
import queue


class Throttle:
    def __init__(self, interval, function, transaction_queue,
        send_dummy=False, dummy=None):
        self.interval = interval
        self.function = function
        self.queue = transaction_queue
        self.send_dummy = send_dummy
        self.dummy = dummy
        self.e = threading.Event()
        self.t = threading.Thread(target=self.throttle)
        self.t.start()

    def throttle(self):
        while not self.e.wait(self.interval):
            if self.send_dummy:
                if self.queue.empty():
                    arg = self.dummy
                else:
                    arg = self.queue.get()
            else:
                arg = self.queue.get()

            threading.Thread(target=self.function, args=arg).start()
```

12.2 Construction of a Data-Embedded Transaction

```
# Convert to base64 for safe transmission
enc_data = base64.b64encode(data)

# Packet: [tube_idx]:[packet_idx]:[packet_content]
packet = f'{tube_idx}:{packet_idx}:{str(enc_data)}'.encode()

preimage, phash = next(self.crypt)

# Keysend record for invoice-free transaction, as well as the data
    carrying record
custom_records = {
    5482373484: preimage,
    9780141036144: packet
}

# The request with the embedded custom records
request = routerrpc.SendPaymentRequest(
    payment_hash=phash,
    amt=1,
    final_cltv_delta=40,
    dest=bytes.fromhex(self.target_pk),
    timeout_seconds=2,
    dest_custom_records=custom_records,
    fee_limit_sat=30,
    no_inflight_updates=True,
    dest_features=[9],
)

for update in self.routerstub.SendPaymentV2(request, metadata=[('
    macaroon', self.macaroon)]):
    print(update)
```

12.3 Key Generation as implemented by LND

```
// generateKey generates a new key for usage in Sphinx packet
// construction/processing based off of the denoted keyType. Within Sphinx
// various keys are used within the same onion packet for padding generation,
// MAC generation, and encryption/decryption.
```

```
func generateKey(keyType string, sharedKey *Hash256) [keyLen]byte {
        mac := hmac.New(sha256.New, []byte(keyType))
        mac.Write(sharedKey[:])
        h := mac.Sum(nil)

        var key [keyLen]byte
        copy(key[:], h[:keyLen])

        return key
}
```

12.4 Throughput Evaluation Script

```
function bench() {
    results=($(curl -so /dev/null -s $1 --proxy 127.0.0.1:8742 -w '%{
        size_download} %{time_total} %{speed_download} %{
        time_starttransfer}' --insecure))
    echo "${results[0]} ${results[1]} ${results[2]} ${results[3]}"
}

function bench_n() {
    s_sum=0
    t_sum=0
    for ((n=0;n<$1;n++))
    do
        res=$(bench $2)
        read -a resarr <<< $res

        time=${resarr[1]}
        speed=${resarr[2]}

        s_sum=$((s_sum + speed))
            t_sum=$((t_sum + time))
        echo $speed $time
        sleep 5
    done

    s_avg=$(bc <<< "scale=0;$s_sum/$1")
    s_readable_avg=$(bytesToHumanReadable $s_avg)
    t_avg=$(bc <<< "scale=0;$t_sum/$1")
```

```
    t_readable_avg=$(bc <<< "scale=2;$t_avg/1000000")
    echo $s_avg bytes per second, $s_readable_avg per second. Average
        duration was $t_readable_avg.
}
bench_n $1 "https://en.wikipedia.org/wiki/Censorship"
sleep 30
bench_n $1 "https://www.bbc.com/"
sleep 30
bench_n $1 "https://github.com"
```

12.5 Evaluation Testbed

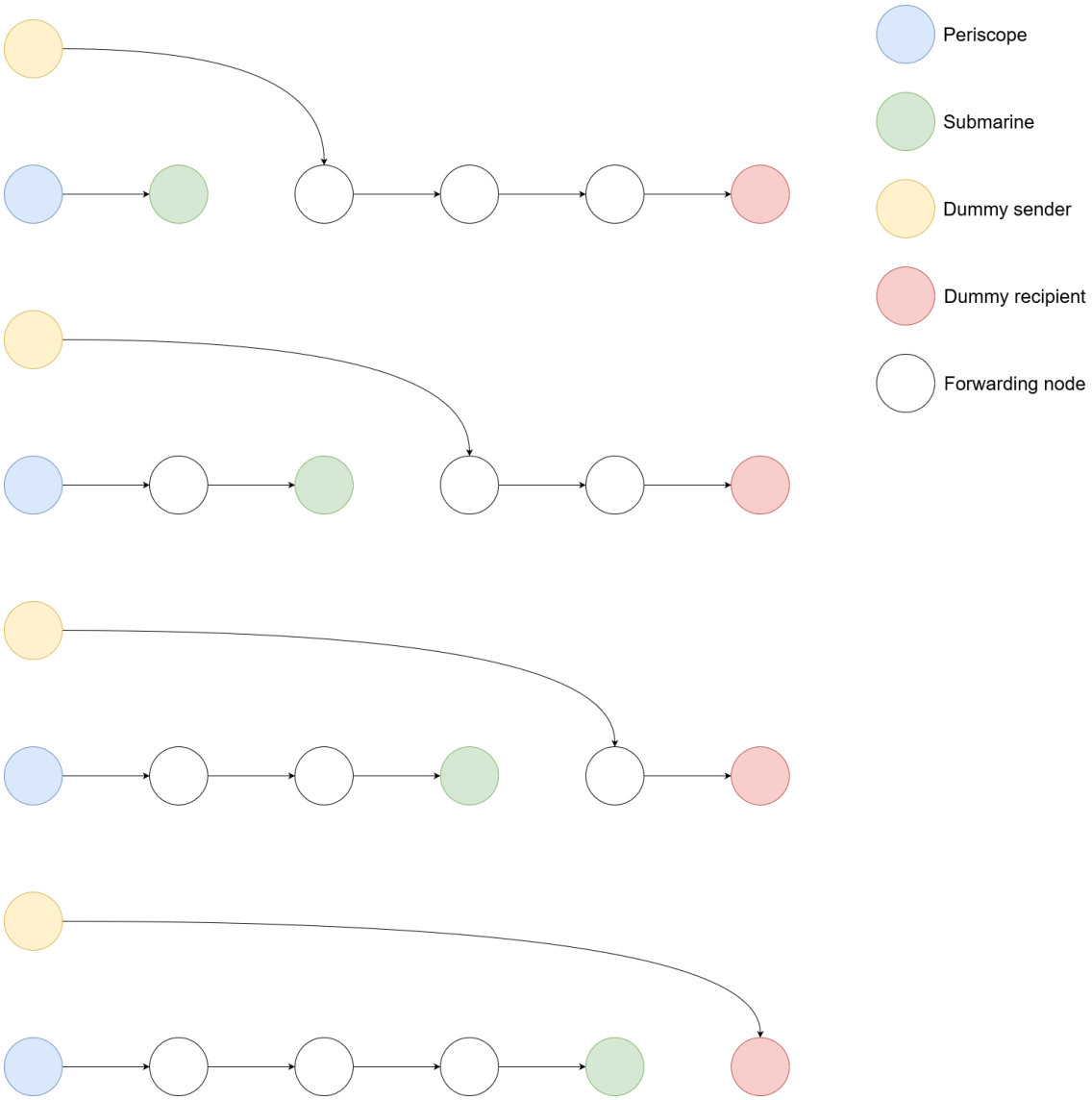


Figure 17: Docker testbed setup to have consistent load during different tests.