University of Zurich^UZH

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Traffic Counting in Mesh Networks

Philip Giryes
Adliswil, Switzerland
Student ID: 19-752-799

Supervisor: Eryk Schiller
Date of Submission: April 21, 2022

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Die rasante Entwicklung der Technologie erfordert Architekturen, die sich an diese hohen Anforderungen anpassen können. Die Verwendung von Mesh-Netzwerken kann eine dynamische und kosteneffiziente Möglichkeit für horizontale Skalierbarkeit bieten. Obwohl Mesh-Netze viele Vorteile haben, verlieren diese Netzwerke die Fähigkeit den Verkehr zuverlässig zu überwachen. Ein Monitoring in Mesh-Netzen wäre für die Netzbetreiber von Vorteil und würde die Opportunitätskosten für die Adaption von Mesh-Netzen, im Vergleich zu herkömmlichen Netzwerk-Topologien, senken. In dieser Arbeit wird mit einem neuen Protokoll versucht einen neuen Ansatz für die Abrechnung des Datenverkehrs zu bieten. Das Cascade Encryption Protocol (CEC) koppelt den Überwachungs- und Datenverkehr, indem es Verschlüsselung zur Einbettung der Routing-Informationen in die Pakete verwendet.
Das CEC-Protokoll wurde in realitätsnahen Umfang auf der Abilene-Topologie simuliert und zur Ermittlung des erzeugten Overheads in kleinerem Umfang auf einer Linien-Topologie getestet. Die Analyse zeigte eine Korrelation zwischen dem Overhead und der Anzahl der Flows bei den Tests im kleinen Umfang. Ausserdem bleibt die Leistung des Protokolls bei in einem voll ausgelasteten Netz flach und nimmt nur langsam ab, wenn die Anzahl der Flows steigen.

The rapid development of technology requires architectures that can adapt to this high demand. Using mesh networks can provide a dynamic and cost-efficient way for horizontal scalability. Although mesh networks have many advantages, these networks lose the ability to monitor traffic reliably. Traffic monitoring on mesh networks would be beneficial for network operators. It would reduce the opportunity cost of adapting mesh networks compared to traditional network topologies. In this thesis, a new protocol will try to provide a new approach to traffic accounting. The Cascade Encryption Protocol (CEC) will couple monitoring- and data traffic utilizing encryption to encapsulate routing information in the packets.
The CEC protocol was evaluated using simulations on a larger scale on the Abilene topology for a real-life-like environment and line topology on a smaller scale to determine created overhead. The analysis showed a correlation between the overhead and the number of flows on the small-scale tests. Furthermore, the protocol's performance stays steady on a fully utilized network and degrades slowly as the number of flows increases.

Acknowledgments

I would like to thank my supervisor, Dr. Eryk Shiller for his assistance and patience in the development of this thesis.

Furthermore, I thank Prof. Dr. Burkhart Stiller head of the Communication System Research Group (CSG), for allowing me to conduct my bachelor thesis under his group.

Lastly, I am grateful to my parents for their moral support during the writing of my thesis.

Contents

Chapter 1

Introduction

1.1 Motivation

The number of Internet of Things devices is estimated to increase globally to 75 billion by 2025. [1]. There is also little indication that the number of devices will decrease soon [2]. With this problem at hand, the need for a scalable and dynamic infrastructure is inherent for networks requiring ad-hoc connections and generally for more traditional broadband networks.

Mesh Networks can conveniently provide horizontal scalability, with less network overhead and less infrastructural dependence [3]. There are multiple large-scale mesh networks in productive use, for example, the Athens Wireless Metropolitan Network (AWMN). The AWMN is a Wireless Mesh Network for extending the broadband reach of the Athenian region with over 2000 nodes [4], and many more [5], [6]. Even though Mesh Networks have their advantages, the mesh topology structure also bears disadvantages compared to the more common star typologies. The collaborative, distributive behavior of the systems makes the traffic less transparent for centralized entities, for example, ISPs and network admins [7].

A solid Traffic Monitoring (TM) scheme for Mesh Networks would lessen the disadvantages, for a small overhead cost, but simultaneously make the advantages of the usage overweigh. Additionally, known structures of traditional networks, such as billing, could be supported based on the statistics gathered on Mesh Gateways. TM could also give some insight into the traffic for governance, congestion resolution, and detection of illegal activities [3]. Overall the benefits would improve the quality of service for the organization- and community-owned networks and the adoption of mesh technology. Research on the topic of Traffic Accounting in mesh networks has been done extensively, but the approaches proposed were oftentimes solutions where the TM acts as a separate entity, where TM itself creates control traffic. This thesis proposes a protocol, that unifies the data- and monitoring- traffic.

1.2 Description of Work

The protocol proposed in this thesis aims to solve this gap in accounting schemes by not trusting third parties, with the help of encryption and a structural approach to leverage nodes to share statistics with the central entity. This approach will bind the routing information of the mesh network to the packet and reduce redundant monitoring traffic generated by the nodes, compared to the constant unicasting of statistics. This protocol will allow the protocol gateway also differentiate between data traffic and forwarding traffic and deterministically recreate the routed path the packet went through in the mesh network. This proposed protocol and other added extensions will be evaluated in a simulated environment, where the performance of handling traffic will be measured and compared to other accounting schemes.

1.3 Thesis Outline

I will first describe the fundamental pieces of my thesis in the second chapter, starting with mesh networks and ending with accounting. Chapter 3 will lay the ground of previous research done until now, and in Chapter 4, I will focus on the high-level view of the proposed CEC protocol and explain in detail how the accounting scheme functions at its core. Furthermore, I will describe the technical intricacies of the implementation of the protocol and describe the infrastructure for the simulation of the system in the following chapter, where I will focus on some network metrics to evaluate the protocol and compare it to the performance of other accounting protocols. In the last chapters, I will present my results, discuss the meaning, and share some insights.

Chapter 2

Background

This chapter tries to lay the foundation for the understanding of the base technologies used in the following chapters. It begins with Mesh Networks and their properties, continues with Traffic Accounting and the existing schemes for traditional networks and finishes off with the GNS3 software, which will play an integral role in the evaluation of the protocol.

2.1 Mesh Networks

Mesh Networks (MN) are networks that, provide a decentralized and dynamic paradigm [8]. Often times there are one or more nodes in the network that act as gateways to the internet or other networks. The other interconnected nodes in the network share access to the gateways and forward the traffic to the nodes, that are not directly connected to the gateways [9]. This can, as mentioned in the introduction, let these networks scale well, by adding more nodes, and the network becomes more resilient and independent of individual nodes [3].

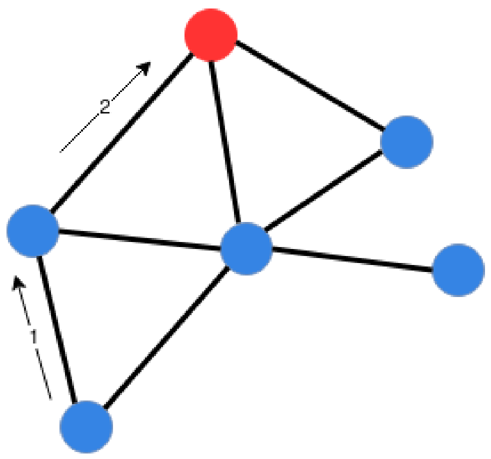


Figure 2.1: Highlevel view on Mesh Networks

To illustrate Mesh Networks with an example, in Figure 2.1, the mesh gateway is marked in red and the mesh nodes are marked in blue. Given that node B, wants to request something from the internet, the underlying routing mechanism routes it to one of node B's neighbors and so forth until it reaches the mesh gateway, which will make the request on behalf of node B. The way from the mesh gateway to the receiver node can be the same route or through another intermediary node, depending on the routing input. This multi-hop structure is the reason, why IP-based traffic monitoring on traditional network architectures will not be applicable for MNs.

2.2 Traffic Accounting

Traffic Accounting[1] or Traffic monitoring is the process of tracking and monitoring network usage, by measuring the amount of transmitted and received data by different devices. This information is used to gain insights into network health and network performance and allocate resources more effectively [10]. Additionally, could the monitoring give insights on potential bottlenecks, low utilization of links, and overall help troubleshoot the network, using traffic engineering [7].
In traditional networks, where the end-devices communicate with the entity in a star-like topology, the data collection usually takes place on the central node, which is owned by the organization. This structure allows an accurate collection of the source and destination of the connected endnotes using NetFlow [11] or IP-based solutions, which are oftentimes router built ins.

2.3 Abilene Topology

Abilene [12] was a network, developed by the internet2 organization, which was meant to provide a network backbone for different research and education purposes. The topology was based on the demographic positions of some larger US cities and the link between the routers achieved a 10 GB/sec. This topology will be the topology used throughout the thesis.

2.4 GNS3

Graphical Network Simulator 3 (GNS3), is a powerful tool for network administrators that allows them to launch network topologies using virtual machines or docker containers [13]. GNS3 provides a realistic software-based testing environment that doesn't require additional hardware to be set up and wired by hand. GNS3 allows you to configure links between nodes, set up routers, hosts, and switches, and spin up all instances simultaneously. The addition of docker functionality in GNS3 has several advantages, such as the ability to easily integrate containers into network topologies. Furthermore, the use of docker containers increases the performance in simulation due to the fact that docker containers are lightweight and require fewer resources than VMs [14]. GNS3 also facilitates

[1]Traffic Accounting, Traffic Monitoring or simply Accounting will be used interchangeably

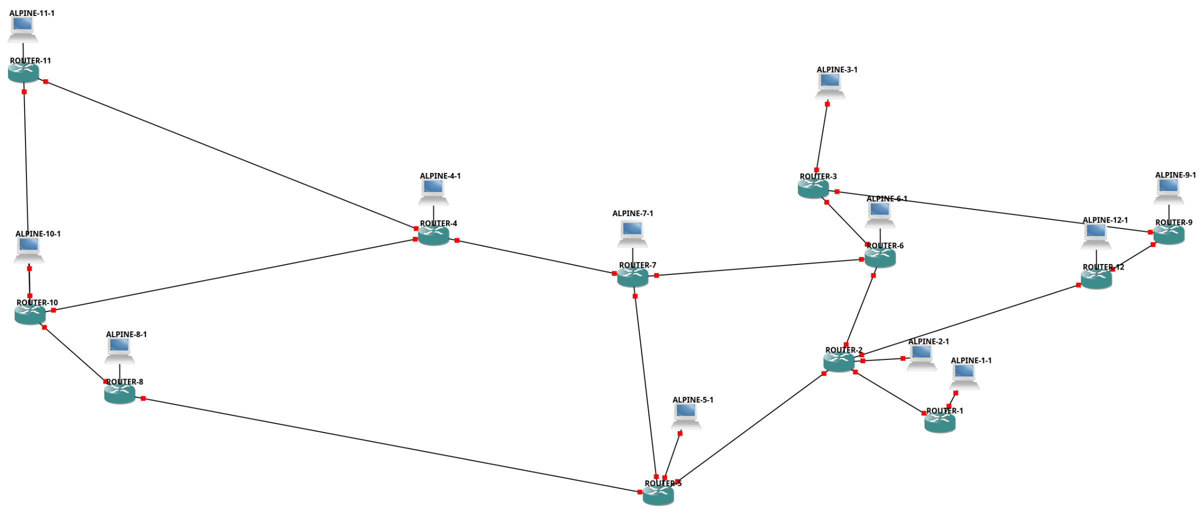
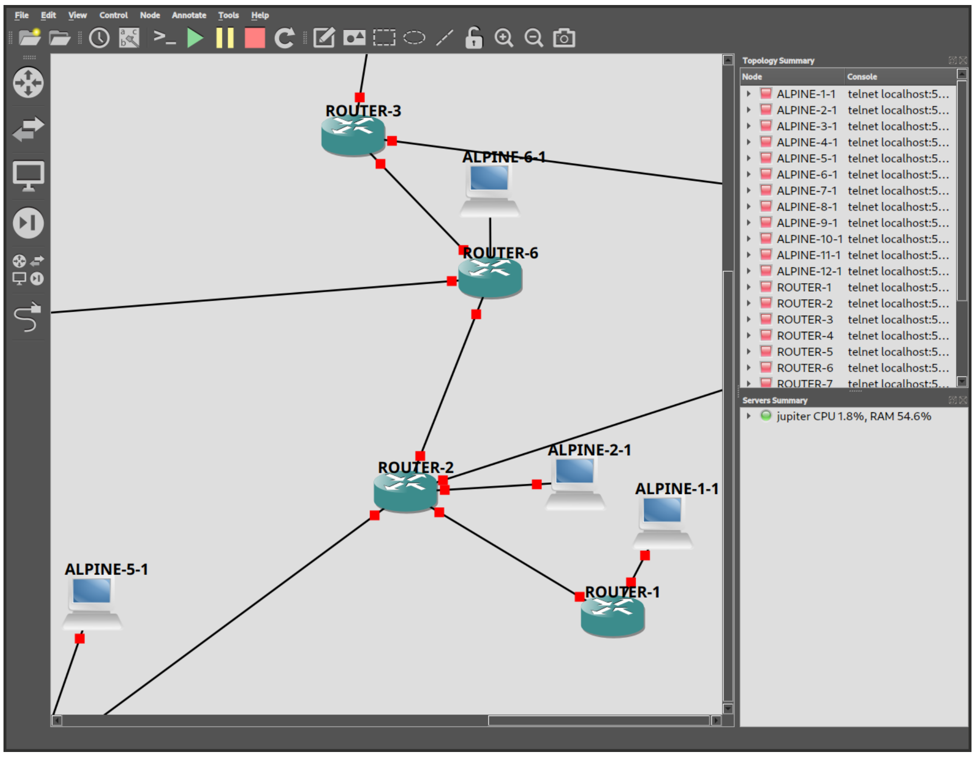


Figure 2.2: Abilene topology in GNS3

troubleshooting with the option to open a shell on running instances, which improves the development workflow, especially in a network environment, where the debugging of processes spans over multiple machines.



Topology Summary

Node	Console
ALPINE-1-1	telnet localhost:5...
ALPINE-2-1	telnet localhost:5...
ALPINE-3-1	telnet localhost:5...
ALPINE-4-1	telnet localhost:5...
ALPINE-5-1	telnet localhost:5...
ALPINE-6-1	telnet localhost:5...
ALPINE-7-1	telnet localhost:5...
ALPINE-8-1	telnet localhost:5...
ALPINE-9-1	telnet localhost:5...
ALPINE-10-1	telnet localhost:5...
ALPINE-11-1	telnet localhost:5...
ALPINE-12-1	telnet localhost:5...
ROUTER-1	telnet localhost:5...
ROUTER-2	telnet localhost:5...
ROUTER-3	telnet localhost:5...
ROUTER-4	telnet localhost:5...
ROUTER-5	telnet localhost:5...
ROUTER-6	telnet localhost:5...
ROUTER-7	telnet localhost:5...

Servers Summary

jupiter CPU 1.8%, RAM 54.6%

Figure 2.3: User-Interface of the GNS3 application

Chapter 3

Related Work

Research was already done on Traffic Monitoring in Mesh Networks, and different approaches have been taken. Most works were done on Wireless Mesh Networks since it has interesting use cases. Regarding accounting, however, the transmission medium takes less precedence and would not hinder the extension of the research to Mesh Networks in general.

Huang [15] et al. proposed an aggregated collection of network statistics of end devices through a Mesh backbone. Huang expanded the definition of network flow with MeshFlows to encapsulate the mesh-relevant information in the records and group similar documents together in one MeshFlow. Furthermore, the MeshRouters build these MeshFlows based on the seen traffic and cache those entries until the Mesh Collector collects the cached data and aggregates and analyses the data to create a system-wide view based on the reconstructed history of the network traffic.

A similar approach to [15] was taken by Cruz et al. [16], where every node caches the hashes of the received and forwarded packets. If a mesh node starts a session at the Base Station, every received packet will be hashed, and the hashes will be sent bundled to the Base Station. Every intermediate node will also hash the received packet and report the hash to the Base Station to claim rewards. The base station keeps track of sessions and compares the hashes to the pre-calculated hashes to analyze bad forwarded traffic and verify the reward-claiming of intermediary nodes.

Cruz [16] and Huang [15] are similar in the sense that both rely on the collection of MeshRouters to keep track of the traffic and then forward the traffic to the Mesh Collector/Base station. However, [16] still adds the ability for the Base station to verify the traffic using hash functions.

Sailhan [17] takes a structural approach to monitor the system with self-configuring clusters. Those clusters are hierarchies that adapt well to the dynamic nature of WMNs and organize and collect the network traffic in a distributed manner. Each cluster has an elected Cluster Head, to which the local nodes will report to. At the same time, the Cluster Heads are responsible for reporting to the Cluster Controller, which will then aggregate the data. The cluster structure creates a distributed handling of statistic collection, which can adapt well to the changing topologies and reduce the monitoring traffic by introducing a hierarchy. Hence the nodes that report to the Cluster Controller are limited.

Gupta [18] analyzed the different approaches of different schemes and proposed a tiered monitoring scheme that combines reactive, statistical, and threshold-based methods. The system configures nodes that satisfy the Minimum-Set-Cover problem to determine the minimum number of Collecter Nodes that would cover every node in the system using only one hop, which is the structural aspect. Collector Nodes are preconfigured with a threshold and reporting frequency to reduce the overhead the monitoring traffic generates. The threshold reduces the traffic necessary to message the Controller Node, and the reporting frequency is an additional parameter to further reduce the overhead by setting delays on reported information.

Frangoudis [19] proposes a system to enable Roaming Capabilities in Wireless Community Networks, where each user can open up his access point for other WMN users and share his bandwidth. An incentive for sharing the bandwidth is when the user wants to profit from the same roaming capabilities and collect his debt for previously issued bandwidth. The proposed Peer-To-Peer Network Confederation Protocol (P2PWNC) allows each roaming end device to request access points (transitively also with Request Repositories) after an established session, where first the node exchanges its receipts for possible requests. The receipts act as a way to monitor the traffic, verify the correctness, and possibly block requests of malicious users. Frangoudis was meant for billing or bandwidth balancing. Nevertheless, this approach could also allow for traffic monitoring since the receipts are an elegant solution to couple monitoring- and data traffic.

Other systems rely on incentives for nodes to share packet-related information with the base station and to compensate for the collaborative effort of forwarding nodes [7] and [16]. The incentives are also argued to be a way to prevent selfish nodes, which only consume traffic from the network, but don't forward any traffic to neighboring nodes. Still, the work of Ben Salem et al. [7] also mentioned that poor incentive systems could act as an "double-edged sword" since it can be an incentive for malicious activity to claim rewards from other nodes.

Even though a lot of research was done on traffic monitoring systems, there seems to be a pattern emerging; namely, the traffic will be recorded and unicasted to the gateway or any other central entity. Some improve the structurally as [15], [16], and [17], and others tried to reduce the messages sent using threshold and other approaches as [18].

[19] took a different approach, which requires receipts as a network communication prerequisite. Essentially the receipt does exchange traffic for statistics. The research seems to be assuming, except [19], that Traffic Monitoring should be a separate entity that also generates monitoring traffic. Little approaches were taken where the traffic information is embedded in the data traffic, which would enforce the sharing of statistics.

Chapter 4

Approach

This chapter will give a high-level overview of the new protocol, the necessary components, and the roles of the different nodes. The focus lies mainly on the accounting scheme. The encryption used, and other technical details will be discussed in the Implementation chapter.

4.1 Protocol Building Blocks

To collect reliable statistics of network-based metrics, e.g., throughput or usage of an MN, the Chain Encryption Protocol (CEC) proposes two necessary components: a Protocol Gateway and a Protocol Node.

The Protocol Gateway (*pg*) acts as the gateway for Internet access and takes the producer role, and the Protocol Node is the consumer and gains the Internet access. Because the traffic necessarily needs to go through the *pg* to access the internet, the traffic is bound to the gateway, which knows the exact amount of MN-outgoing traffic. The *pg* is assumed to be a trusted party, mainly because the gateway holds the keys of all the nodes; it is also essential. The *pg* also holds an account of the activity that is going on in the mesh network. With this account, the gateway can measure traffic and theoretically also engineer the traffic. There can be more than one Protocol gateway, which the Protocol could also support, but for simplicity's sake, the assumption is that there is just one Protocol Gateway.

The Protocol Node (*pn*), as mentioned before, benefits from Internet access. The traffic the nodes generate is not always the traffic they generate/request, the *pn* may additionally act as an intermediary node for the request of other nodes as an effect of the collaborative nature of MNs. This is one of the main problems the protocol needs to account for and hence cleanly separate the forwarding traffic from the generated traffic to not skew the statistics. The Protocol Nodes could join and exit the Network in a hop-on-hop-off manner, but since the focus lies more on the core protocol itself, a static topology is assumed without the loss of generality.

The Cascade Encryption (CE) used is symmetric encryption because of the lower overhead during runtime, contrary to asymmetric encryption. The core of the protocol is Cascade Encryption, which allows the traffic to stay private for other nodes), detects routing differences, and plays an integral role in the accounting of the traffic. The type of encryption is not relevant and is also protocol independent, but more about that in the Implementation chapter.

Real-life scenario:

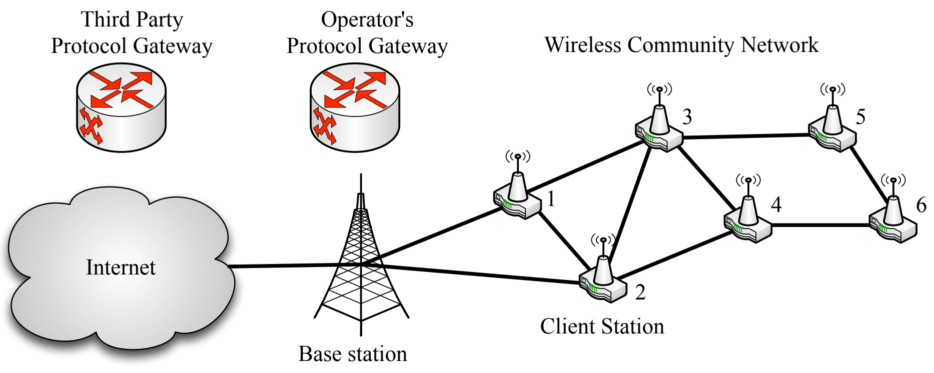


Figure 4.1: Possible scenario of a Wireless Mesh Network

As illustrated in Fig 4.1 the protocol nodes would be nodes (3,4,5,6) and the protocol gateway would be out of (1, 2) node nr. 2. If for instance, node 4 would make a request either node 2 or node 3 would act as an intermediate node and forward the request to the gateway. This illustration will be used as an example, the topology shown will be used, throughout this thesis without the loss of generality.

The Accounting of the CEC protocol consists of two parts depending on whether the traffic is from the node to the gateway (upstream) or from the protocol gateway to the node (downstream). The structure of both procedures is similar, except for some minor differences on the downstream part. The following section will focus on each protocol part in more detail.

4.1.1 Upstream Accounting

As mentioned in the Accounting section, the upstream protocol transmits a message from its current position to the Protocol Gateway, with the help of other intermediate forwarding nodes. The packet source will encrypt the packet with its encryption, in the same way, each forwarding node will encrypt the packet upon receipt.

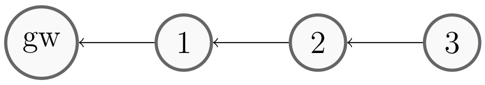


Figure 4.2: Simple upstream

Based on the simple case in Fig. 4.1.1 there can be a situation where the source is node 3, and there are two intermediate nodes (2 and 1), which act as a forwarding node before the message gets to the protocol gateway *pg*.

The process of the upstream can be described within 3 Stages:

Initial encryption at the source

Node 3 encrypts the request (data block) with its secret key and prepends its header and id to the message. The packet will then be forwarded, from the routing proposed neighbor to node 2, since the *pg* is multiple hops away. This step creates the initial encryption layer of the cascade encryption, which will primarily keep the data private from other nodes.

Resulting Message $\mathbf{u_1}$:

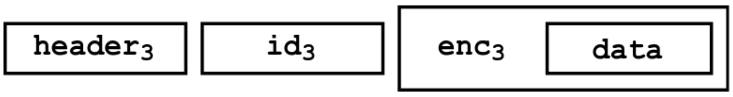


Figure 4.3: Message u_1 send from node 3 to node 2

Cascade Encryption at the intermediate nodes

Each forwarding node encrypts the package with its key and forwards it to the next node. Additionally, the header of the forwarding node will be prepended to the package, to maintain the networking information needed in the lower layers. Moreover, the id of the node will be added to facilitate the Accounting (in the last step of the Upstream Protocol).

In the example, node 2 receives the package from the message u_1 and encrypts the packet with the key_2 and adds id_3 and the header. Also, node 2 can't differentiate if the packet received originated in node 3 or was only forwarded by the node. The only information node 2 has been, that the sender one hop away was node 3 and the destination is the gateway[1].

Resulting Message $\mathbf{u_2}$:

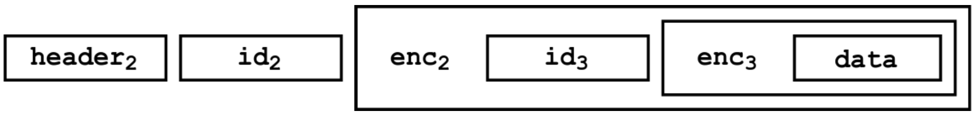


Figure 4.4: Message u_2 send from node 2

[1]inference could be drawn by measuring the packet size, but this could be prevented by padded packets

Message u_2 will then be forwarded to node 1, and the same procedure is done as mentioned for message u_2, resulting in message u_3, which will then be sent to the *pg*, because the gateway is a direct neighbor of node 1.

Resulting Message $\mathbf{u_3}$:

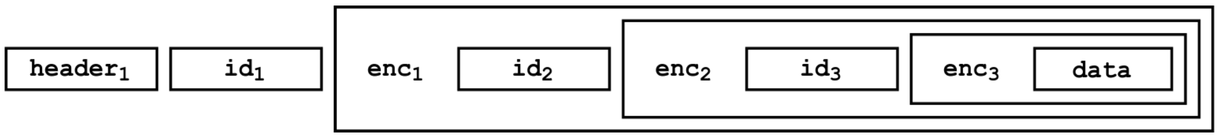


Figure 4.5: Message u_3 send from node 1

Accounting Process at Protocol Gateway

The gateway will start stripping off the layers of encryption, on the one hand, to get the data packet, and on the other hand to extract the route the packet took for the Accounting. The *pg* will know which packet to use, based on the node id sent in clear text with the packet. In the example above, the gateway receives the packet from node 1 and sees the id number 1, therefore the decryption requires the key 1. This could be easily achieved since the gateway has all the keys. The *pg* encounters keys 1, 2, and 3 and infers that the route went first through node 2 and then node 1 (and node 3 is the source node).

4.1.2 Downstream Accounting

The Downstream is analogous to the Upstream Protocol, with the addition of tokens. To expand on the example given before but on the downstream side consider Fig. 4.1.2.

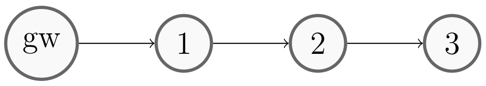


Figure 4.6: Simple downstream

In this case, the packet will take the same route back as it came. The process of the downstream can be explained in the following four stages:

Initial encryption at the protocol gateway

Since the downstream starts at the protocol gateway, the *pg* will encrypt the data packet similar as in the Upstream and construct the message d_1, but also add a token unique for this package for later identification of the packet, and also encrypt it and prepend it to the data-packet.

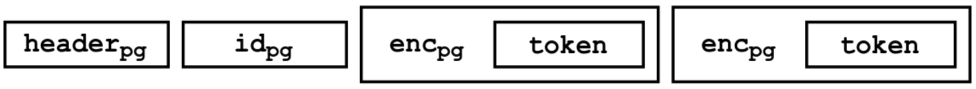


Figure 4.7: Message d_1 send from the pg

Resulting Message d$_1$

Cascade encryption at the intermediate node

The cascade encryption on the downstream adds an encryption layer to the data packets and the token. Continuing the example, d_2 will be prepared on node 1, and d_3 will be constructed on node 2 before the packet reaches its final destination: node 3.

Resulting Message d$_2$

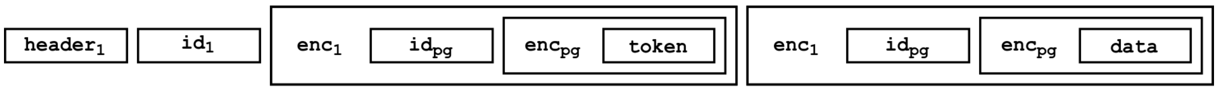


Figure 4.8: Message d_2 send from node 1

Resulting Message d$_3$:

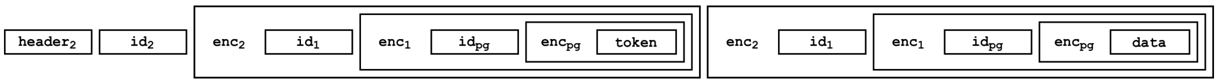


Figure 4.9: Resulting Message d_3 send from node 2

Requests for a temporal key

The received message d_3 will be unreadable for node 3, hence the missing decryption keys. Node 3 needs to extract the encrypted token packet and send it to the protocol gateway to request the necessary keys. The token acts as proof, more on that in the next subsection.

Resulting Message d$_4$:

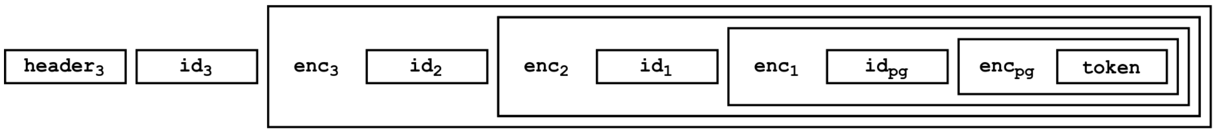


Figure 4.10: Message $u1$ send from node 3

Accounting Process at Protocol Gateway

The message d_4 is received at the *pg*, and the Accounting Process starts. First, the layers of the encryption are stripped, and the ids of the layers are noted to determine the route. Second, to verify that d_4 originated from the source node and isn't just an arbitrary node requesting decryption keys, the first layer of encryption must match the source node. Second, the clear-text token should match the token created at the *pg*, if they match, indicating that the request belongs to a legitimate communication session and proof for the reception of the gateway response started at d_1. Third, the node depends on the token's sending to read the data; the token acts as a hostage for the keys.

Sends temporal keys

After verifying and accounting using the d_4, the gateway generates the correctly configured keys necessary for encrypting the packet at node 3 and sends the packet encrypted with the encryption of the receiving node. Intermediary nodes will transparently forward the packet without reiterating the downstream protocol.

Resulting Message $\mathbf{d_5}$:

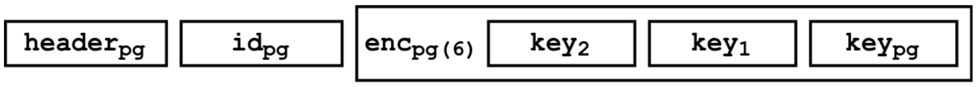


Figure 4.11: Message d_3 send from node 3

Node 3 will be able to peel off the layers of encryption and access the clear-text data, thus completing the downstream interaction.

Chapter 5

Implementation

This chapter will start with a top-level view of the different components and will gradually focus on more details of the base CEC-Protocol and the different variations. The end of the chapter describes the testing environment and paves the way for the evaluation chapter.

5.1 Network Configuration

Topology of the network is the Abilene topology already extensively described in the background chapter.

Routers are docker containers where the routing and configurations are already installed. The routers will run the CEC Protocol Router application. The router docker containers need to be prebuilt before launching the topology using GNS3.

Hosts are docker containers with CEC Protocol Userspace applications installed. The actively running application is decided on container build time. Like the router containers, also the host containers need to be prebuilt.

Routing used on the routers is OSPF using the FRRouting implementation. Without any application running on the routers and hosts, every machine and router is pingable, even when the machines are not on the same network (which creates the mesh view on the network).

5.2 Protocol Userspace

This section will be a high-level documentation of the userspace software's relevant components and responsibilities. The detailed functionality of the protocol was described in Chapter 4, and these elements can characterize the technical structure:

Tunnel interface is configured with an point-to-point connection to the protocol gateway. Because of the tunnel connections, the overall topology on the top layer looks like a star, where every *pn* is directly connected to the *pg*. This virtual network interface also allows our userspace program to write and read packets, which facilitates the manipulation of packets [20].

Tunnel thread listens on the tunnel interface for data packets coming from the userspace or the socket Thread. When data is found, the tunnel thread encapsulates it into a packet and sends it to the destination.

Socket thread waits on the socket port for incoming packets, which will be managed according to their type, and response messages will be sent. Moreover, the socket thread could utilize the tunnel interface to offload some work to the tunnel thread.

Resource object is a global object containing information relevant to the two different threads. Packet queues, keys, send-timers, and their respective mutex are included. The Resource object is a global object and can be accessed by every software component at runtime.

Multihost support is provided by establishing one running CEC Userspace instance per *pg* - *pn* connection. The ports and tunnel interfaces are configured and passed to the program as command-line arguments. Each running instance needs its own port and tunnel interface. Otherwise, conflicts occur.

Inter Process Communication (IPC) Support is necessary to configure userspace interfaces in the testing environment simultaneously. Since the userspace program is a server and does not return when started, the program can not communicate with the outside when the environment is initialized. This could induce some race conditions or runtime errors; an example would be the configuration of interfaces that haven't been created yet. The implementation of communicating through a Linux named pipe helps to prevent this issue.

Packet Handler is a class used in the Socket Thread. The responsibility of this class is to parse received aggregated packets and their subpackets. After recognizing a packet type, the binary representation will be serialized into the proper packet object, and the handler takes actions based on the object type.

Symmetric Encryption used is key-based, meaning that a password for the node, a salt an iteration count is necessary to generate a key [21]. By increasing the counter new keys could be generated, without compromising any information. The encryption standard used is not relevant, in a sense, if the standard becomes obsolete, it can be swapped out as long as the interfaces stay the same.

These components are also a part of the Piggyback- and Router-Extension; if no differences are mentioned in the corresponding sections, the same structure is assumed.

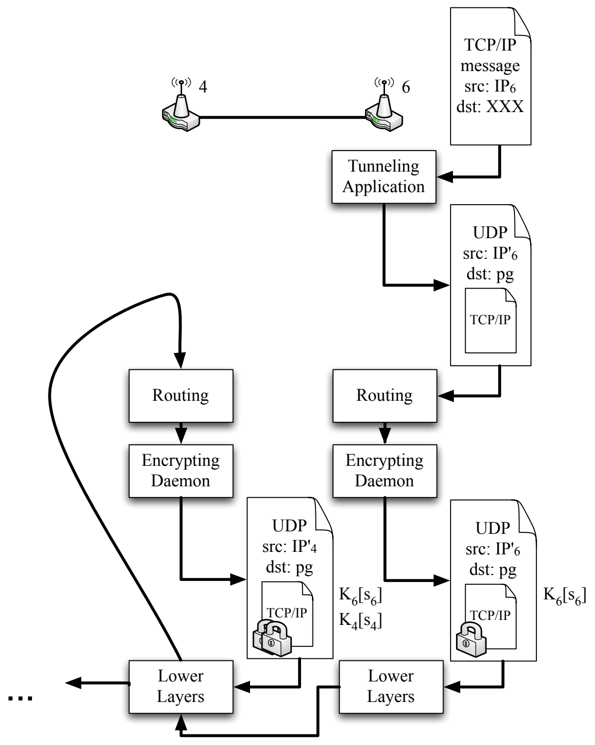


Figure 5.1: Architecture of the protocol

5.3 Protocol Router

The CEC Protocol Router or Filter is a single-threaded application with only one purpose: to encrypt forwarding traffic with its key, as seen in the approach chapter, with messages using Cascade Encryption on the down- and upstream.
The filter application utilizes the libnetfilter-queue in combination with an iptables rule to access packets from the kernel space, modify them (encryption), and reinject modified packets again [22]. The only exceptions are key packets in message d_5, which are not modified by the Filter application and instead forwarded as is. In Fig. 5.1 the Router Application is labeled as "Encryption Daemon".

5.4 Piggyback Extension

The Piggyback Extension on the CEC-Protocol uses a technical solution to reduce messages exchanged, especially on the downstream, where, from a high-level-view, are three messages[1] exchanged between the *pg* and the *pn* opposed to the upstream where only one message[2] is required. The Piggyback back solution uses the open TCP sliding window of at most 10 ms to hold some packets back, waiting for packets with similar destinations,

[1]messages are: (d_1-d_3), d_4 and d_5
[2]message is (u_1-u_3)

append to the other packets, and create an aggregated packet. Hence this will save at least one request.
In the best case, multiple packets could be piggybacked on one another, saving the additional requests and reducing the monitoring/session traffic. If there are only a few packet losses in the system and the piggyback mechanism times out a lot, the system will fall back to the regular scheme; for each keys-request, each response will be sent individually with a minimal delay. However, the problem is when piggybacked packets get lost, and the multiple packages will be lost together, amplifying the loss rate.

5.5 Testing environment

5.5.1 Setup

To start the testing environment, the containers must be built, the build_docker_containers.sh, the script has two building functions, one for the router and one for the host. The functions facilitate the execution of multistage docker build and are set up to reduce the overall building time by relying on cached entries.
The GNS3 environment can be set up using the Abilene.gns3 file. GNS3 automatically updates the docker containers in the configuration file to the latest build. When GNS3 spins up all containers correctly, the routing and the userspace applications should be up and running (for routers and hosts).

5.5.2 Tests

To test the network performance, iperf3 [23] is a versatile tool that allows bandwidth testing on various protocols, including other metrics such as packet loss, latency, etc. Each test will be started, based on the parsed topology, with different parameters preset, for example, the connection pairs, ports, and the test duration. First, the iperf servers will be started on the *pg* and the *pn*s. Then the client connections will be created gradually to prevent the issue of failing tests due to iperf3-clients connecting to not fully started servers. Each test between the *pg* and the *pn* will be bidirectional, meaning that a server is always started on both nodes and a client started on both. One sample configuration of parameters of a test instance can be seen in table 5.1.

From the technical perspective, the iperf3 instances are configured and launched from a Python script[3] on the host machine utilizing the docker SDK [24] for running the bash scripts on the respective containers. After completing all the instances, some aggregation and visualization scripts will process the raw data further to the results presented in the next chapter.

[3]more details to the processes is in the software documentation written.

Setting	Range	Value
Client	$\{1 - 12\}$	4
Server	$\{1 - 12\}$	7
Port	$\{1 - 6000\}$	4004
Protocol	UDP/TCP	UDP
Parallel	$\mathbb{N}$	2
Bitrate	$\{0.0 - 10.0\}$m	1.0m
Duration	$\mathbb{N}$	60
Packet length	$\mathbb{N}$	1000

Table 5.1: Example iperf3 settings and values

5.6 Flow Configuration

Before running the network tests, an essential component is to predefine the flows, composed of source, destination, and bandwidth allocation. In this section, two models are presented, the gravity model and the greedy model, though only the latter will be used in the evaluation.

5.6.1 Gravity Model

This model is based on the work of [25], where the author combined the idea for the gravity model with the traffic simulation on Abilene topologies. The model is based on some Newtonian principles, where the connected nodes and the demographic distance influence the predicted traffic. This is a synthetic traffic model, which is statistically similar to real Abilene traffic. From this model, a $N \times N$ matrix could be derived to determine the bandwidth for the flows from a node i to a node j. One sample distribution provided by this model can be found in Fig. 6.1.

5.6.2 Greedy Model

The greedy model is a much simpler traffic model than the model presented before because it only uses the highest possible bandwidth values without degradation of service of the least performing process. Each tested flow uses the same bandwidth for sending the testing requests, unlike the gravity model, where each flow has a different bandwidth depending on the source and destination. The advantage of this system is that the model's parameters are easier to tweak. Additionally, the model helps to determine the performance of different systems in relation to the least performing; if this process is of interest, gives some valuable insight. An example configuration of the greedy model can be found under Fig. 6.1.

Chapter 6

Evaluation

This chapter will evaluate the different schemes based on the section 5.5. The different schemes are Cruz-Accounting, CEC-Accounting, and CEC with the Piggyback-Extension. The network without accounting schemes will be measured alongside the other schemes to get a sense of the overhead that these protocols create, using the topology and tools described in section 5.1. The overhead will be the delta between the network performance without configuration subtracted from the scheme's performance.

6.1 Notes on Comparability

Since in the CEC-based schemes, the traffic is either issued by some node and forwarded to the gateway or vice versa, the topology looks like a star, as mentioned before. The problem lies in the Cruz scheme and the bare-bone network without configured accounting schemes. Both are suitable for mesh typologies and allow flows with a node as a source and another as a destination. For the sake of comparability of the different configurations, the same flows will be used on all schemes (resp. non-schemes), even though the flows will be a subset of the possible flows of the schemes as mentioned above.

6.2 Traffic distribution

A visual representation can help the understanding of the gravity traffic model described in 5.6.1. The model generated the traffic matrix based on the underlying topology and the distribution of end devices on the routers. Fig 6.1 is a heat map representing a traffic matrix using 24 flows (or 48 connections). The darker blue color indicates a lower number of allocated traffic, and the dark red color a high amount of traffic. The matrix is symmetric like the traffic matrix and is also zero-indexed. As an example, the coordinate (0, 6) is the respective connection from node 1 to node 7 (the gateway).

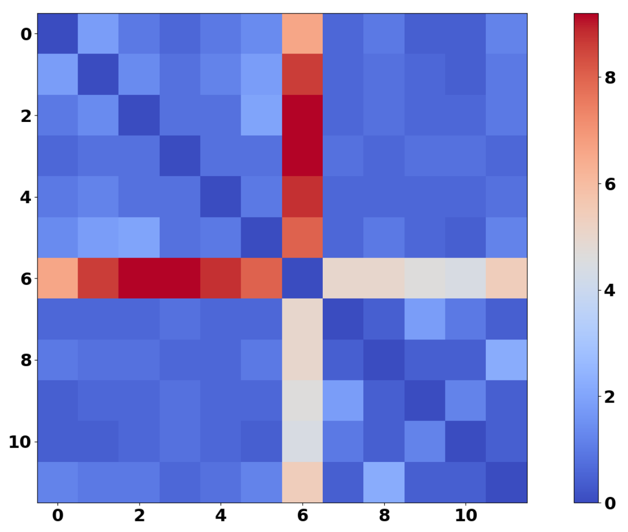


Figure 6.1: Heatmap of gravity model

The diagonal of the matrix has no traffic since that corresponds to the node's traffic to itself. The horizontal and vertical are more reddish since node seven lies very central (demographically), and the connected nodes are very well connected; thus, higher traffic is expected in this region. The matrix's four quadrants are heterogeneous and weighted based on the demographical properties.

Even though a matrix of a total $N \times N$ matrix was generated for the simulation, only an $(N-1) \times 2$ matrix is necessary, as already pointed out in the section 5.6.1. Only line 6 would be necessary as input because it encapsulates all the traffic from and to the gateway.

6.3 Global Performance

To measure the real-life network performance of the protocol, 6 to 24 flows were set up to test the network. It measures a real-life-like environment with parallel running network requests, where each process could compete with other processes on bandwidth and processing power. The simulation environment used was described in this section 5.5.

Throughput/Goodput was tested on four configurations: the bare-bone system, CEC application, CEC with the Piggybacking extension, and Cruz implementation described in [16]. The model that was used is the Greedy model described in 5.6.2. The baseline comparison was CEC because it was expected to have the lowest performance. The parameters were chosen so that every test passes all configurations and yields the most throughput. Table 6.1 shows the used configuration on all global tests.

Flows	6	12	18	24
Bandwidth	1.4	0.9	0.5	0.35

Table 6.1: Configuration for the global test environment

Table 6.1 shows the configuration for the CEC application that will be used for all others. Important to note that the bandwidth per flow will be lower for more flows in the system. The more flows compete with other flows on the links, the more messages will get dropped, and more traffic loss will occur on a fully utilized link. Failing tests due to overloaded links would let tests fail, skew the statistics, and make the results irreproducible.

6.3.1 Throughput

Figure 6.2 depicts the four different test results based on the number of flows. Surprisingly, the Cruz scheme's performance is similar to the bare-bone system, where only the routing is an active component. The bare-bone system acts as a practical and theoretical maximum of throughput that the CEC protocol could achieve and acts as a control group.

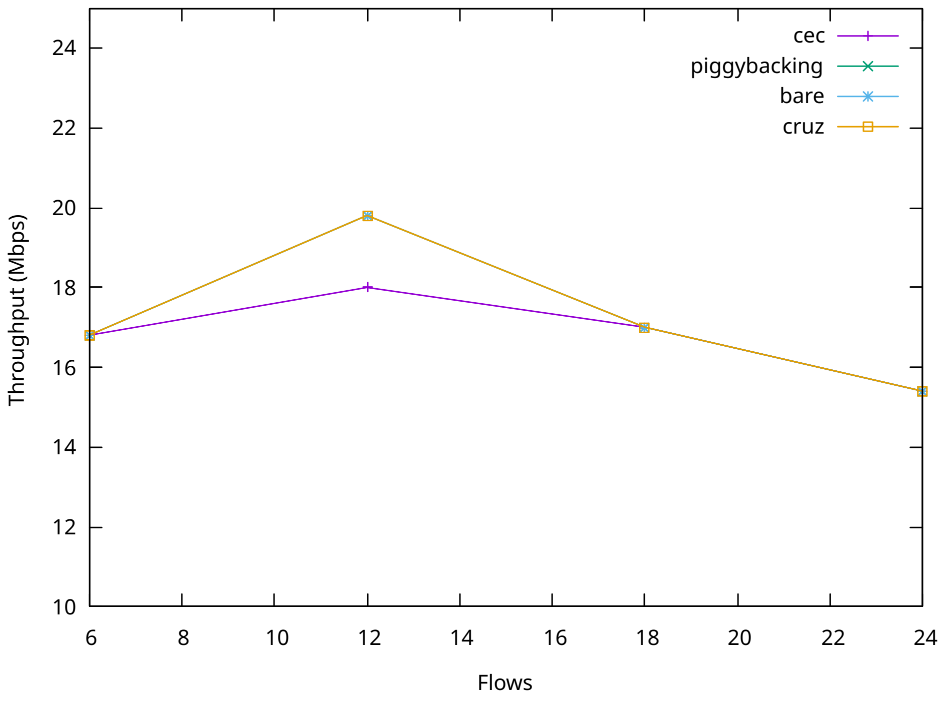


Figure 6.2: throughput of the network simulation

A trend on all the schemes is the uptick in throughput with 12 flows configured. This could be explained based on the "economy of scale" effect, and each additional flow adds more productivity to the system than it does compete with other flows. Hence from 6 flows to 12 flows, only new active machines will be added before running multiple processes on the machines.

The throughput for the Piggybacking extension and the CEC without any extension is identical. There seems to be no observable advantage of Piggybacking on the throughput side. Nevertheless, their identical performance helps to analyze the difference in goodput later in this chapter.

On a range of flows tested, the CEC application had a steady amount of throughput and slightly declined on the increasing number of flows. The difference between the bare-bone system and the CEC-based solutions is higher on a lower node/flow count and insignificant on a higher one.

6.3.2 Goodput

The data on the global goodput is derived from the throughput test, and the configurations' parameters are the same. The data gathered on the goodput was created by the same test instance as the throughput, which means that the goodput on, e.g., flow 12 corresponds to the throughput of flow 12 in 6.2.

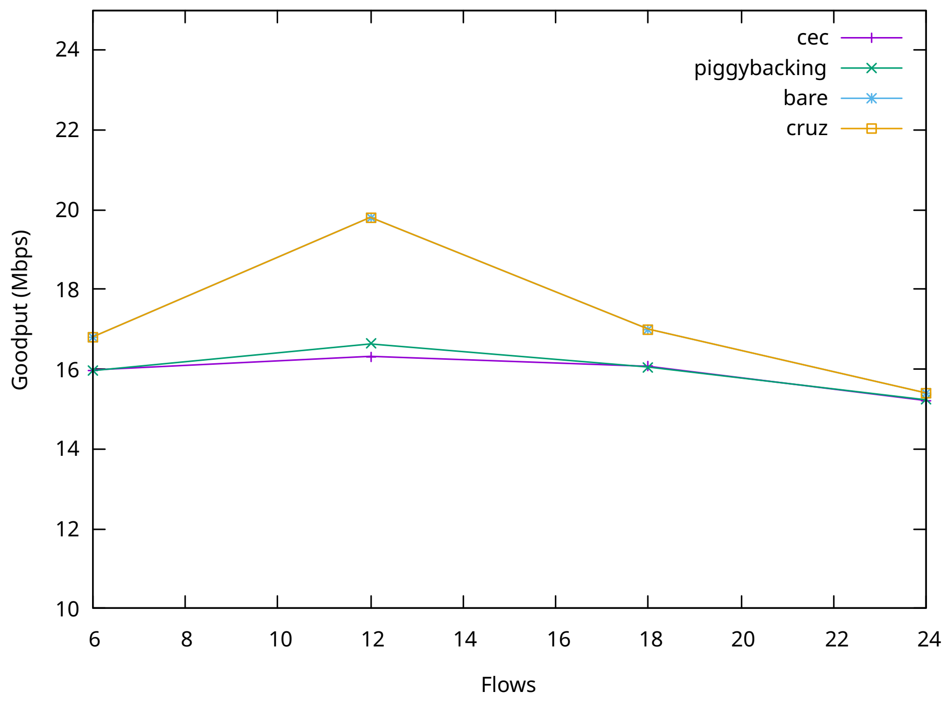


Figure 6.3: Goodput of the network simulation

Also, here, surprisingly, the goodput of the Cruz scheme is almost identical to the performance of the bare scheme. The bare-bone system has only a slight observable overhead and represents the upper bound of maximal achievable goodput, analogously as 6.2.

Even though Piggybacking was minimally performing better on flow number 12, the conclusion is that piggybacking doesn't add any significant improvements on the tests done with UDP; this could look different on TCP since there are messages exchanged (ACK messages). This result is an exemplary visual representation of the fallback mechanism of Piggybacking, which was described in 5.4. The difference between the goodput- and throughput- line of the CEC applications is maximally 1 MB/s and only a little decline over the number of flows.

Summarizing, the results presented in 6.2 and 6.3 are very astonishing; even though the comparison to Cruz and Piggybacking was reduced, possibly by the reasons mentioned, the CEC protocol seems to behave very reliably for the tested flows and also, the overhead observed seemed to be more of fixed, than of variable nature. Furthermore, the convergence of the goodput of the CEC application to the bare-bone performance on increasing flows beats all expectations. Still, a further evaluation of a broader range of flows needs to be conducted to support this observation fully.

6.4 Upstream Performance

The tests conducted in this section and in the following section 6.5 are done on a modified Abilene topology, where the node path resembles a line of a maximum of 10 nodes. The throughput measurement relating to the number of hops was done on three different schemes, and each test was repeated ten times on the same configuration. The Cruz scheme was left out of this evaluation since the scheme in the global scope was not differentiated from the bare system. Each result entry consists of the population's mean and standard deviation.

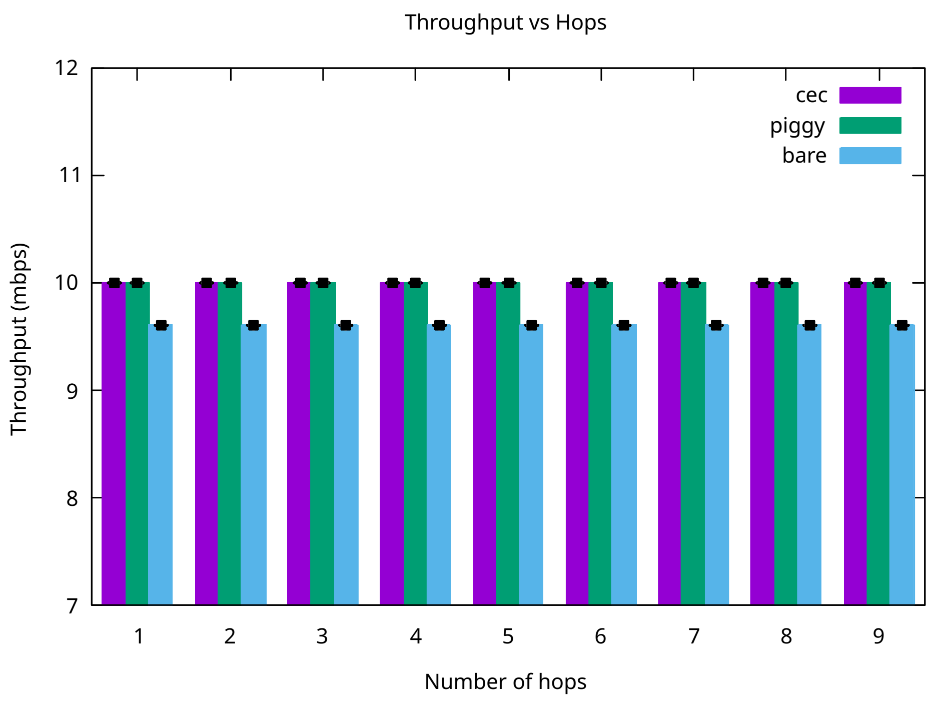


Figure 6.4: throughput on upstream

As shown in Fig 6.4, the throughput is at the max value for all schemes, with little deviation from the mean. The max value is 10 Mbps because the underlying interfaces were configured only to support this maximum bandwidth. A possible explanation is that since there is only one test running at a time, as opposed to the environment of the global network tests, the throughput is not influenced by external variables. That is of benefit since it creates a clean environment for the experiments. The more exciting part is the goodput evaluation in 6.5 and how the goodput relates to these results.

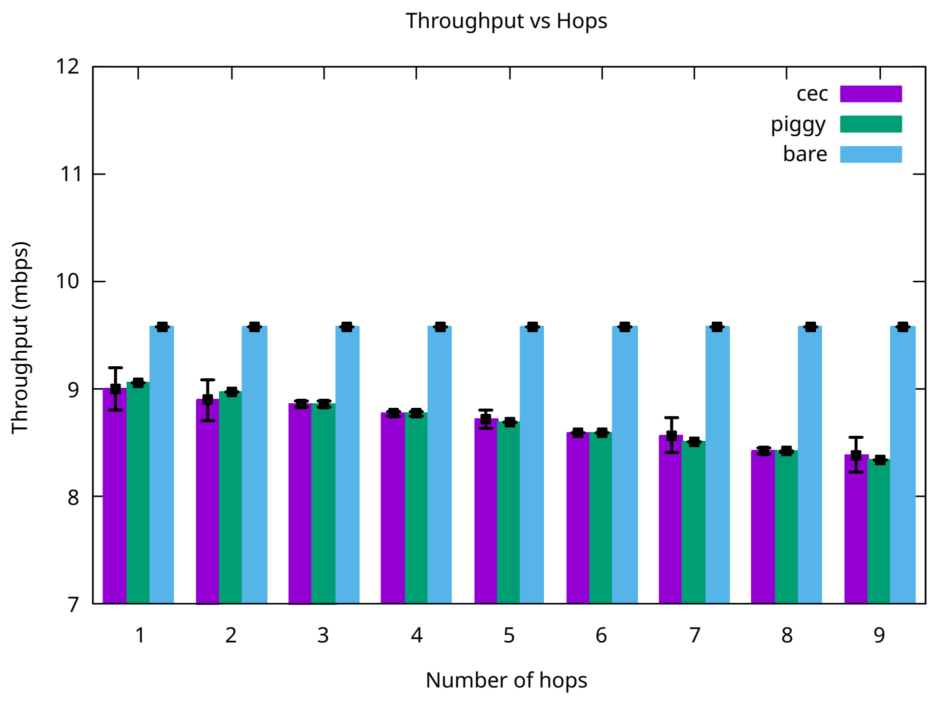


Figure 6.5: goodput on upstream

Figure 6.5 shows a constant value for the bare scheme, with little deviation from the mean. CEC and the piggybacking extension show similar results with little to no difference, with slightly more deviation on the side of the base CEC application.

The inverse dependence of goodput versus the number of hops is also interesting. The performance degradation on multiple hops was expected, though the results show that the difference between one hop and nine hops is only max 0.5 Mb/s with a little deviation of the mean. Also important to note is that on one hop, the difference between the throughput and goodput is 1 Mb/s, which indicates the fixed overhead of the application, and the difference of 0.5 Mb/s difference between 1 and 9 hops is the variable overhead.

6.5 Downstream Performance

The downstream was measured analogously to the upstream, only that the gateway connects to the server instances on the end nodes. The differentiation between downstream and upstream is essential since more messages are exchanged on the downstream protocol. Based on the theoretical structure, it is also expected to perform lower than the upstream.

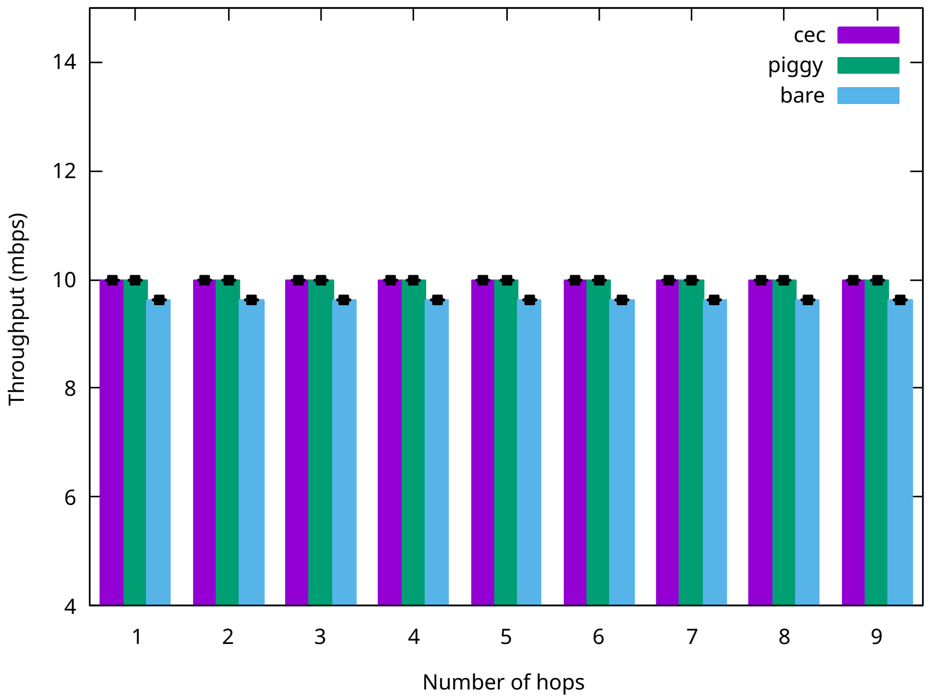


Figure 6.6: throughput on downstream

Figure 6.6 shows the mean of throughput achieved over all tests per configuration. This test result looks almost identical to Figure 6.4. As mentioned, the protocol has no problems translating the 10 Mb/s bandwidth and fully utilizing the underlying interface's maximum capacity.

Results in 6.7 show a different picture than the goodput results on the upstream. The downstream performance seems arbitrary and not linked to the hop count. For example, 1 and 8 hops show a meager goodput rate, whereas 5 and 6 show the highest rate. The results are difficult to interpret with this graph alone.

Also, the difference between the piggybacking extension and the standard CEC application seems minimal to non-existent on the local scale. The reasoning could be similar to the upstream, namely that the piggybacking is not tested in the right environment for the optimization to be visible. For Piggybacking, a TCP-based test could be more suitable.

To summarize the results of the local scope, one could say that the throughput reaches the maximum capacity on the upstream as on the downstream, and even not depending on the number of flows. In general, the results seemed to have low deviations from the mean, which increased the significance of the results. The downstream was challenging to interpret, and no obvious relation could be found between the variables. Though fascinating insight was derived from the upstream since a fixed overhead of 1Mb/s and variable overhead of a maximum of 0.5 Mb/s could be determined.

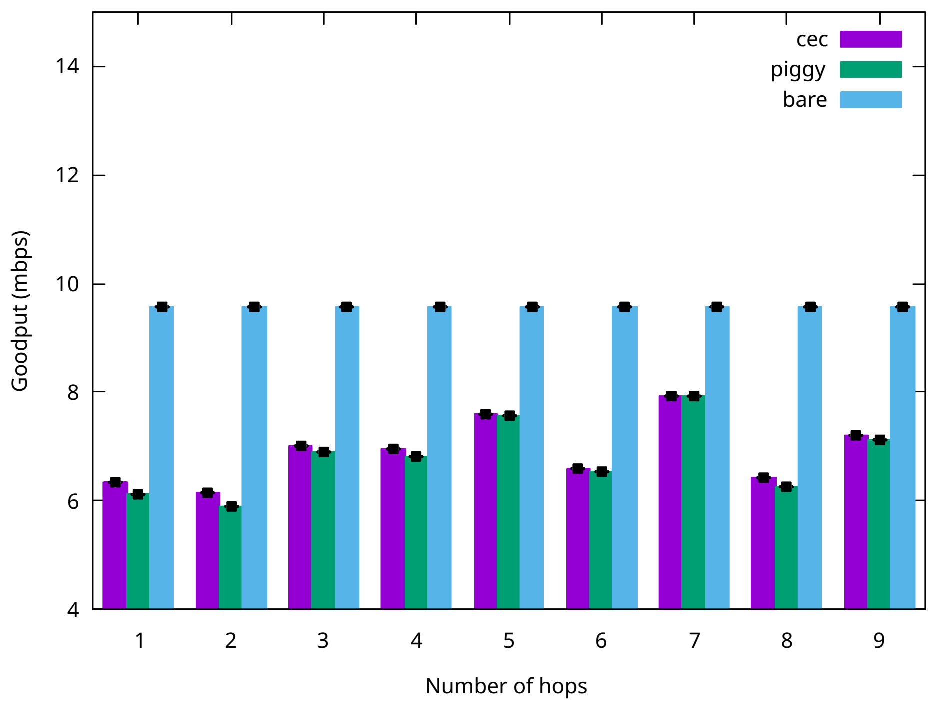


Figure 6.7: goodput on downstream

Chapter 7

Discussion

This chapter will discuss the relevance, scope, and meaning of the result presented in the last chapter and put the results into context for interpretation. The results will also be critically analyzed from a technical, theoretical, and logical perspective.

The results on the local level[1] of the protocol, which only focussed on the performance of the upstream or downstream, are significant. Since the results with ten hops are a worst-case scenario in the Abilene topology. The tests were conducted using blocked and suspended links, forming a line. This scenario is improbable in the Abilene topology and acts as an upper boundary. Also, the experiments conducted on the local scope are done in a very focused manner, which on the one hand, is an environment with only a few parameters that could influence the results. On the other hand, these local experiments give no insights into how the protocol behaves with a higher load on multiple links. Results helped to see the relation between flow number and overhead in the upstream. Furthermore, the CEC protocol‘s throughput seems to be on the local level independent of the number of flows.

A real-life simulation of the protocol can be observed on the global results[2], which gives insight into the protocol‘s performance with an increasing load and compares the CEC application with another scheme as Cruz. 6 - 24 flows are relatively low, but this is all the machine that simulated the network could support CPU-wise. High flows (above 24) would evoke the termination of some necessary application when the system was running on 100% of the CPU. Especially the filter on the router of the gateway was killed sometimes, making all the experiments fail because they could not reach the gateway anymore.

The gravity traffic model mentioned above (5.6.1) could not give a distribution congruent to the limitations of the iperf level. Some tests created a too-high link load, which would let different packets get lost; therefore, other tests failed. So the tweaking of the model parameters was not enough to be supported by the testing environment, so a switch to the greedy model was necessary. The greedy model is suitable for comparing multiple other schemes about one variable. Though the greedy model also has weaknesses, some schemes

[1]sections 6.4, 6.5

[2]section 6.3

could hypothetically support more bandwidth per node, which will not be reflected in the model results. The schemes of the global scope could have been performing on a fraction of the possible bandwidth.

The simulation on the global scope also yielded some exciting results, considering the number of flows given. Compared to the bare system, the protocol‘s steady performance and goodput‘s slow degradation were clearly shown. Moreover, the goodput and throughput of the protocol seemed to converge to the values of the bare system with a rising number of flows for the tested flow range.

The Cruz scheme implementation needs some revaluation since the performance of Cruz was optimal. This contradicts the results by [16], meaning that the implementation is not trustworthy enough to draw conclusions and comparisons. Even if the implementation of the Cruz scheme was correct and would outperform the CEC application, the comparison will be only weak because the CEC protocol has more functionality embedded. For example, the processing time needed by the Cruz scheme to verify the correctness and recreate the packet history is not accounted for but inbuilt into the CEC application. Nevertheless, the comparison is necessary, even if the approaches differ. Additionally, the Cruz scheme was not evaluated on the local scope, assuming the performance would not differ from the base system. This assumption could be proven wrong if the Cruz scheme was implemented correctly.

Furthermore, there are still many aspects to this topic that make it challenging to evaluate. From the model‘s decision to the technical implementation and the simulation parameters. For the scope of this thesis, it would be too much to evaluate the protocol holistically and weigh the differences in results. Nevertheless, an effort was made to highlight the validity or limitations of the results. It is also important to note that this thesis does not contain all the reflections and decisions made, except one example of the model selection and consideration.

Chapter 8

Summary and Conclusions

This thesis proposed a traffic monitoring scheme based on the encapsulation of routing information to the packets with the help of encryption. The network performance of the scheme was analyzed and compared to other schemes and configurations. Although the results were unexpected, several conclusions could be drawn based on the work done. Some interesting insights into the protocol and its behavior in different environments could be derived, and sometimes the results beat the intuition. The stable maximum throughput upstream and downstream suggests that some parts of the protocol have already reached optimality. Furthermore, the convergence to the optimal performance in goodput/throughput with higher flow counts provides some interesting conclusions and provokes some questions for further evaluation.

This thesis still does not cover some areas and could be expanded in future work. The areas presented are technical, algorithmic, and architectural possible improvements that could, in future work, extend the work done on this thesis.

Userspace Architecture

Since the router application is single-threaded, this could influence the result that the protocol can practically achieve. Especially the gateway router is a significant bottleneck because every incoming and outgoing traffic must necessarily pass through this router. Minimizing this bottleneck and increasing the traffic handled on the gateway router or all the routers could improve the performance and remove some technical overhead. Additionally, some tests measuring the processing time of the userspace applications could be invaluable.

Experiment Design

Since the CPU was a limitation for running more than 24 flows in the system, evaluating the same experiments with more computing power would yield more flows and converge to more real-life scenarios. With a higher flow count, the proposed improvement of the global tests, mentioned in chapter 7, could be implemented. This would extend the claim maid on the performance with a higher flow count.

In addition, a possible replacement of the iperf3 tool for software that is similar but more reliable in a stressed environment. The simultaneous deployment of iperf3 instances was sometimes problematic, and the results were often flaky.

Routing-aware extension

A possible improvement in the downstream protocol is to minimize the messages exchanged. A routing-aware version of the protocol could help use some additional and already available information to improve the performance further. In the core, the Routing Extension's Accounting assumes that a packet will take the same route as before; if the routing has changed, the destination node will signal the gateway since it can not read the encrypted packet (because the nodes encrypted the packet differently). In the best case, this solution could save two messages downstream and make the downstream similar to the upstream if the routing stays constant. In the worst case, where the routing changes, the protocol will fall back to the standard procedure and perform similarly to the pure CEC application. The solution would be an algorithmic improvement and could reduce the overhead dramatically.

Security

Not only to see this protocol under normal or stressed circumstances but to see how the protocol handles common attacks on mesh networks, for example, flooding or denial of service attacks. Additionally, evaluating the security of the protocol specification itself based on modern cryptographic methods would prove the correctness and validate the specification. Security is crucial, mainly because the monitoring- and data traffic are coupled.

Final thoughts

With the information of chapter 3 in the back mind, the protocol takes a novel approach and combines different ideas. The picture the results paint should also motivate us to expand on this protocol further. Overall the CEC protocol and its extension seem to be promising and should be, in my opinion, further looked into.

Bibliography

[1] D. Georgiev, *Internet of things statistics, facts & predictions [2023's update]*, review42, Mar. 2023. [Online]. Available: `https://review42.com/resources/internet-of-things-stats/` (visited on 04/18/2023).

[2] E. Schiller, "Landscape of iot security", eng, *Computer Science Review*, 2022, ISSN: 1574-0137.

[3] M. L. Sichitiu, "Wireless mesh networks: Opportunities and challenges", in *Proceedings of World Wireless Congress*, vol. 2, 2005, p. 21.

[4] *Awmn wind - wireless nodes database*, wind.awmn.net. [Online]. Available: `https://wind.awmn.net/` (visited on 04/08/2023).

[5] I. Dalmau, *What is guifi.net? | guifi.net*, guifi.net, Jun. 2009. [Online]. Available: `https://guifi.net/en/what_is_guifinet` (visited on 04/08/2023).

[6] "Worum geht's? - freifunk.net", *freifunk.net*, [Online]. Available: `https://freifunk.net/worum-geht-es/` (visited on 04/08/2023).

[7] N. B. Salem, L. Buttyán, J.-P. Hubaux, and M. Jakobsson, "A charging and rewarding scheme for packet forwarding in multi-hop cellular networks", in *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking & Computing*, ser. MobiHoc '03, Annapolis, Maryland, USA: Association for Computing Machinery, 2003, pp. 13–24, ISBN: 1581136846. DOI: `10.1145/778415.778418`. [Online]. Available: `https://doi.org/10.1145/778415.778418`.

[8] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: A survey", *Computer Networks*, vol. 47, no. 4, pp. 445–487, Mar. 2005. DOI: `10.1016/j.comnet.2004.12.001`. [Online]. Available: `https://doi.org/10.1016%5C%2Fj.comnet.2004.12.001`.

[9] D. Benyamina, A. Hafid, and M. Gendreau, "Wireless mesh networks design - a survey", *IEEE Communications Surveys &;c Tutorials*, vol. 14, no. 2, pp. 299–310, 2012. DOI: `10.1109/surv.2011.042711.00007`. [Online]. Available: `https://doi.org/10.1109%5C%2Fsurv.2011.042711.00007`.

[10] J. Svoboda, I. Ghafir, V. Prenosil, *et al.*, "Network monitoring approaches: An overview", *Int J Adv Comput Netw Secur*, vol. 5, no. 2, pp. 88–93, 2015.

[11] *Cisco ios netflow*, Cisco, Jul. 2017. [Online]. Available: `https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html`.

[12] *Abilene backbone network*, web.archive.org, Dec. 2007. [Online]. Available: `https://web.archive.org/web/20071213210730/http://abilene.internet2.edu/` (visited on 04/15/2023).

[13] S. Yuen, *Getting started with gns3 | gns3 documentation*, mother.github.io, Jul. 2020. [Online]. Available: `https://docs.gns3.com/docs/`.

[14] S. Yuen, *Docker support in gns3 | gns3 documentation*, mother.github.io, Jul. 2020. [Online]. Available: `https://docs.gns3.com/docs/emulators/docker-support-in-gns3` (visited on 04/08/2023).

[15] F. Huang, Y. Yang, and L. He, "A flow-based network monitoring framework for wireless mesh networks", *IEEE Wireless Communications*, vol. 14, no. 5, pp. 48–55, Oct. 2007. DOI: `10.1109/mwc.2007.4396942`. [Online]. Available: `https://doi.org/10.1109/Fmwc.2007.4396942`.

[16] E. R. Cruz, D. Camara, and H. C. Guardia, "Providing billing support in wimax mesh networks", in *2009 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications*, IEEE, 2009, pp. 161–166.

[17] F. Sailhan, L. Fallon, K. Quinn, *et al.*, "Wireless mesh network monitoring: Design, implementation and experiments", in *2007 IEEE Globecom Workshops*, 2007, pp. 1–6. DOI: `10.1109/GLOCOMW.2007.4437816`.

[18] D. Gupta, P. Mohapatra, and C.-N. Chuah, "Efficient monitoring in wireless mesh networks: Overheads and accuracy trade-offs", in *2008 5th IEEE International Conference on Mobile Ad Hoc and Sensor Systems*, 2008, pp. 13–23. DOI: `10.1109/MAHSS.2008.4660026`.

[19] P. A. Frangoudis, G. C. Polyzos, and V. P. Kemerlis, "Wireless community networks: An alternative approach for nomadic broadband network access", *IEEE Communications Magazine*, vol. 49, no. 5, pp. 206–213, 2011.

[20] M. Krasnyansky and F. Thiel, *Universal tun/tap device driver*, 2002. [Online]. Available: `https://www.kernel.org/doc/Documentation/networking/tuntap.txt` (visited on 04/08/2023).

[21] B. Kaliski, *Password-based cryptography specification*, Ietf.org, Sep. 2000. [Online]. Available: `https://www.ietf.org/rfc/rfc2898.txt` (visited on 04/21/2023).

[22] *The netfilter.org "libnetfilter_queue" project*, 2021. [Online]. Available: `https://wiki.nftables.org/wiki-nftables/index.php/What%5C_is%5C_nftables%5C%3Fi` (visited on 04/08/2023).

[23] *What is iperf / iperf3 ?*, iperf.fr. [Online]. Available: `https://iperf.fr/` (visited on 04/08/2023).

[24] *Docker sdk for python*, docker-py.readthedocs.io, Nov. 2016. [Online]. Available: `https://docker-py.readthedocs.io/en/stable/index.html` (visited on 04/15/2023).

[25] M. Roughan, "Simplifying the synthesis of internet traffic matrices", *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 93–96, 2005.

List of Figures

List of Tables

Appendix A

Installation Guidelines

Extensive documentation on the configuration, building, and running of the simulation is available on a private GitHub repository, to which the CSG members have access.

Appendix B

Contents of the ZIP Archive

1. Thesis (PDF)
2. Thesis as Latex source (ZIP)
3. Source code of the protocol and the repositories for the simulations
4. Midterm presentation slides (PDF)
5. Datasets and charts (ZIP)