HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

COMPUTER SCIENCE

IFS INSTITUTE FOR
SOFTWARE

0-RTT Protocol in Golang

Term Project

Department of Computer Science
University of Applied Science Rapperswil

Fall Term 2019

Author(s):	Thomas Elsensohn, Nicola Stalder
Advisor:	Dr. Thomas Bocek
Project Partner:	IFS
External Co-Examiner:	-
Internal Co-Examiner:	-

Contents

Part I
Project Report

1 Introduction and Overview

ATP is a 0-RTT network protocol with reliability features and built-in asymmetric encryption. It is intended to be similar in features to TCP, while avoiding some of its drawbacks.

While ATP is a general purpose protocol, it is especially useful for developers dealing with peer-to-peer transfer problems, due to its inherent capability of sending and receiving simultaneously.

1.1 Related Work

In large parts, ATP's feature set is inspired by protocols like QUIC and KCP, and those familiar with either protocol will see clear similarities. Both protocols were in fact under consideration for usage in TomP2P. However, neither proved to be applicable for a number of reasons. QUIC for example uses HTTP3 headers and is over-engineered for the intended purpose [1], while KCP is poorly documented and not properly standardized [2].

2 Results

2.1 Current State

In its current state, ATP is fully capable of communicating with another endpoint with full asymmetric encryption following an initial key exchange. The implemented ARQ protocol ensures all data is delivered to the user in the order it was sent, performing retransmissions when data loss is detected. [3] [4]

Overall, most of the proposed feature were implemented. There are however still a few missing features and limitations that will be addressed either in a future project or independently.

2.2 Missing Features

As mentioned, not everything could be fit within the limited time frame of this term project. For some of these features, it was clear right at the start that their implementation would need to be out-scoped. Others however had to be out-scoped due to unforeseen complexity in other features or miscalculation during the early phases of this project. Some of these features will be listed bellow and may serve as reference for future implementation.

NAT Traversal: The most prominent missing feature is ATP's planned ability to traverse networks using NAT to remap their IP address range. The proposed method to achieve this is NAT hole punching, as it still the most popular method in TCP and UDP networks [4]. This feature was never planned to be scoped within this term project. The time and resources required to successfully implement it did not suffice, especially since no one in the development team hat prior experience in this field.

Slow Start and Congestion Control: TCP's and other protocol's ability to throttle its transmission speed according to the current network state [4] was originally planned to be part of this term project. Due to some of the problems mentioned before, and described in greater detail further down, this feature had to be taken out-of-scope. For the time being. This feature however should not require much more resources to implement, as many of the systems that are needed for it to work are already in place.

RTT Measurement: This feature was proposed relatively late as hypothetical solution to retransmission timeout problems. With it in place, the RTT can be measured for each connection, adjusting timeout values dynamically according to the distance between two endpoints. This can also be applied to flow/congestion control to get an ever more accurate measure of how well the network is performing. As this idea was brought up late during development, its implementation was out-scoped for the time being.

Property Object: As of right now, ATP is not built to be configured by users. To address this inflexibility, a property object was proposed that can be passed upon creation of an ATP socket. All values not specifically set by the user is assigned a sensible default value that should fit in most situations. This feature was deemed relatively unimportant and easily implemented should it suddenly be needed. As such, it was out-scoped.

2.3 Causes

Planning for ATP was done relatively loosely and kept agile to adapt to the customers needs. This was done mostly due to lack of time for a lengthy planning period in which all features could be determined categorized. This however also led to some miscalculations and unnecessary work that could have been avoided with a more thorough planning.

Most of this can be attributed to the implementation of ATP's ARQ protocol. In an initial step, it was determined to implement the simpler but less efficient go-back-N-ARQ. This was done in order to quickly develop a working MVP that can then be improved and extended. However, this ARQ implementation turned out to be short-lived, as it became clear that selective-repeat-ARQ still needed to be implemented. Furthermore, the latter proved to be far more complicated than anticipated, and required more time than what was planned.

2.4 Conclusions

While many of the planned features were implemented, much work is still needed before ATP is fully ready to be used in production environment. These include, but are not limited to, everything mentioned in the section "Missing Features".

That being said, the delivered result is a solid start that can easily be improved on to fully develop an excellent network protocol specially tailored for P2P networks. In a future project that aims to further improve ATP, this document may be used to guide development towards the final product and avoid some of the pitfalls that slowed development in this first phase.

Part II
Specification

3 Introduction

3.1 Product Scope

The main goal of ATP is to develop a new network protocol that is similar in features to TCP and QUIC, while at the same time cutting down overhead and complexity. These main features include the implementation a reliable transfer using a variation of the ARQ protocol, an easy to use interface and eventually secure connection establishment with as little round trips as possible.

3.2 User Characteristics

ATP was designed with peer-to-peer transfer in mind, hence the implementation of a two-way-connection interface, which will be described in greater detail later on. Getting started with this protocol's interface should be as simple as possible. To achieve this, a quick start guide will be provided, as well as extended usage possibilities. The code itself will be held to the documentation standards of Go. This should allow for any Go developer to quickly start implementing peer-to-peer connections without consultation of lengthy tutorials.

4 Requirements and Features

ATP is not a typical software product, but a library for other developers. As a result, normal functional and non-functional requirements are hard to write or sometimes even indistinguishable from each other. Because of this, the following chapter instead describes ATP's features as a transport protocol, and how it aims to address common problems in network communications and the important differences to other, similar protocols.

4.1 Header

One of the main goals of ATP is to keep its overhead small. This also includes a small header size. In an initial version, it will only consist of

- Data offset
- Flags
- Sequence number

with a total size of 6 bytes. This size will grow in the future, for example to introduce a dynamic window.

4.2 Connection Establishment

ATP sockets will always act as sender and receiver simultaneously. Connection is established over UDP sockets by default, however the infrastructure will be setup in a way that these two-way connection could be replaced with any other protocol. Unlike TCP, there is no initial handshake when establishing a connection (aside from a DH key exchange). Instead, upon defining an endpoint, data exchange can immediately begin.

4.3 Quality of Service

To ensure an optimal usage of network bandwidth, ATP will implement multiple QoS mechanism to increase efficiency without overexerting resources.

4.3.1 Congestion- and Flow-Control

Flow control: To prevent a more powerful sender to overwhelm a weaker receiver, a sliding window will be introduced that will prevent packets from being sent if too many are still on the wire (not acknowledged). This will be directly integrated into ATP's implementation of the ARQ protocol, described in detail later in this document. To allow adjustment to the window size, the receiver will be able to communicate its capacity to the sender, which will allow the latter to throttle its transmission speed up or down as necessary.

Congestion control: Network congestion is generally inevitable, and not within direct control of the developer [4]. Nonetheless, it has to be dealt with. To do this, ATP will implement measures that allow throttling up or down its transmission speed according to how well the connection performs. If, for example, many segments start getting lost, ATP will gradually slow down its retransmission speed until data is reliably reaching its destination again. Afterwards, transmission speed will be increased until problems arise again, or the limit set by the flow control is reached.

4.3.2 ARQ

The main QoS mechanism used will be the ARQ protocol, which ensures that data is received in the same order as it was originally sent. There are two main variations of this protocol. These are:

Go-Back-N-ARQ: This variant of the ARQ protocol is generally a more simpler approach than other solutions. As it received out-of-order segments, the protocol will simply start dropping segments and stop acknowledging new ones. Instead, all further ACKs will include the last in-order sequence number, as well as the next expected one. As previously stated, this is a relatively simply approach and only requires considerations for a few edge cases. However, this simplicity is the cause for many potential segment drops and duplicated data sent over the wire. [4]

Selective Repeat ARQ: This is a more complex solution. Whenever an out-of-order segment is received, it is buffered instead of dropped. Every ordered segment will be immediately passed to the reader and removed from the buffer. On receiving a segment, a positive ACK is sent. The ACK

encodes a bitmap combined with the sequence number of the first slot in the bitmap. The receiver of the ACK will then resend all segments missing from said bitmap. [4]

Conclusion: To make data transmission as efficient as possible, ATP will implement selective repeat ARQ, in order to minimize segment duplication.

4.4 Security

To make ATP a viable alternative to TCP/QUIC, some form of encryption protocol will need to be implemented. The de facto standard at the time of writing is the AES-GCM block cipher, paired with a Diffie-Hellman key exchange. Implementing these algorithms from scratch is neither necessary nor sensible. Instead, ATP will use the Noise Protocol Framework. This framework boasts numerous features and enables building a security protocol tailored to ATP's specifications. [5] The first goal is to implement a protocol that adheres to the aforementioned de facto standards. Advanced authentication mechanism are possible as well, but are not within the current scope of this project.

4.5 NAT Traversal

Since most potential clients are protected by a firewall or connected to a router using NAT, ATP will need to implement some form of NAT traversal to connect to two clients through these protections. This feature is currently out-of-scope for this project, as it will require significant resources to implement a good solution that will work with most networks. Thus, this feature will be implemented at a later date once all other requirements are fulfilled.

5 Domain Analysis

5.1 Diagram

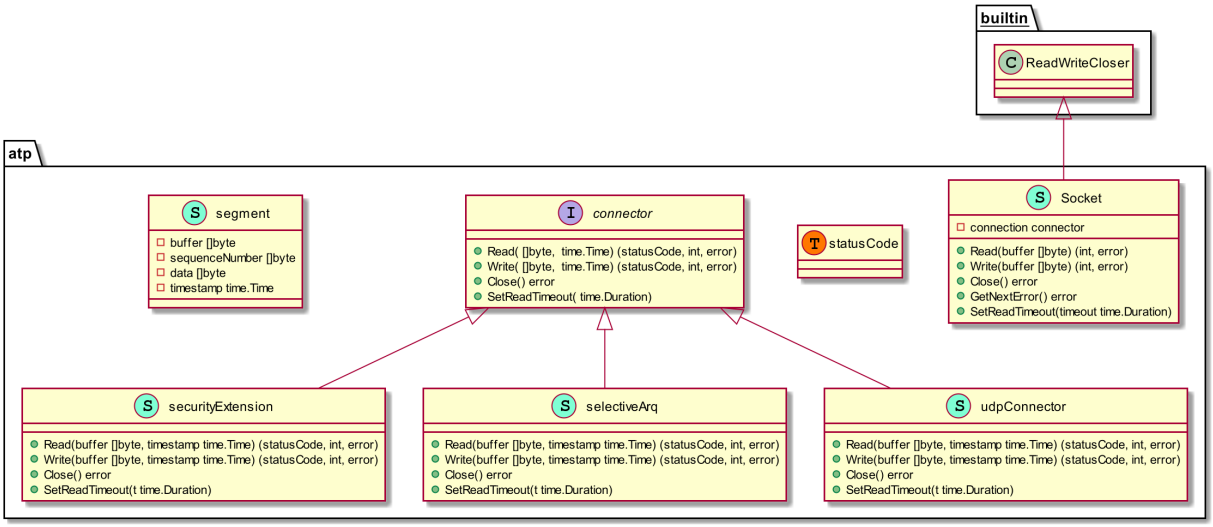

Figure 1: Domain model diagram

5.2 Concepts

5.2.1 Socket

The socket will be the main construct users interact with. All connections are established through these sockets, after providing ATP with the necessary endpoint information. Through this socket, users can also listen for errors that occur at times that can't be passed down properly. The underlying connection will consist of a combination of components, each with their own specific responsibility.

5.2.2 Extensions

ATP uses the decorator pattern to build its connections. The following three components are the current main features, however the entire protocol can be extended by simply injecting a new extension wherever it is needed. This concept makes properly testing ATP significantly easier and enables quick extensibility.

5.2.3 selectiveArq

An implementation of the selective repeat ARQ protocol. It wraps all incoming data into segments to ensure it arrives in the correct order. It's also responsible for ATP's flow and congestion control.

5.2.4 securityExtension

This extension performs all security related tasks, such as key exchange and date encryption and decryption. It uses the Noise Protocol Framework, it could however be switched out for any newly developed extension.

5.2.5 udpConnector

This extension maintains ATP's two-way UDP connection, transferring all data it is passed to the remote endpoint, and reading from it when prompted.

6 Requirements Analysis

By nature, all network protocol have a large amount of edge cases that all need to be addressed. These special cases stem from various problems with network connectivity or lack of reliability. This section aims to specify as many as possible to illustrate how these problems can be dealt with.

6.1 Segment Transmission

6.1.1 Two-way segment, no loss

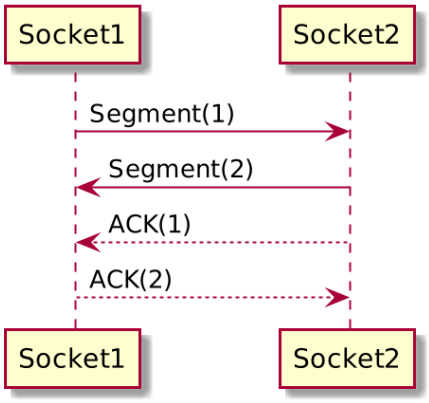

Figure 2: Two-way segment, no loss

Since ATP is a two-way protocol, this case is the usual connection between the two endpoints. Note that the order of all segments arriving is not guaranteed, as opposed to what the diagram shows. This is the best-case scenario, and also the baseline ATP. Where possible, cases deviating from this should not be noticed by whoever uses this protocol, aside from events such as timeouts or a complete connection loss. In such cases, passing down errors accordingly is inevitable.

6.1.2 Singular data segment lost

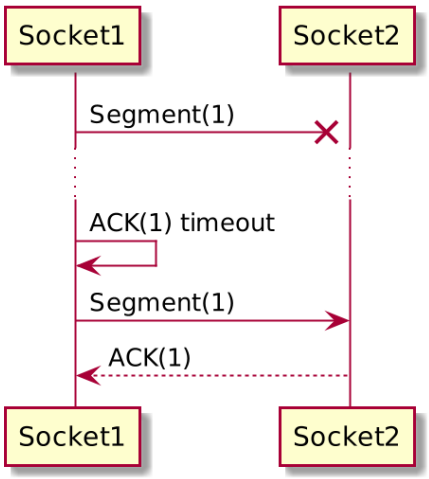

Figure 3: Singular data segment lost

Since UDP packets can always get lost on the way, ATP needs to be able to deal with such events. In the case above, the first segment sent is lost somewhere on the link, which keeps the receiving socket from sending an appropriate ACK segment. For cases like this, the segment will eventually time out, causing the lost packet to be sent again with the same sequence number.

6.1.3 Data segment lost at any point

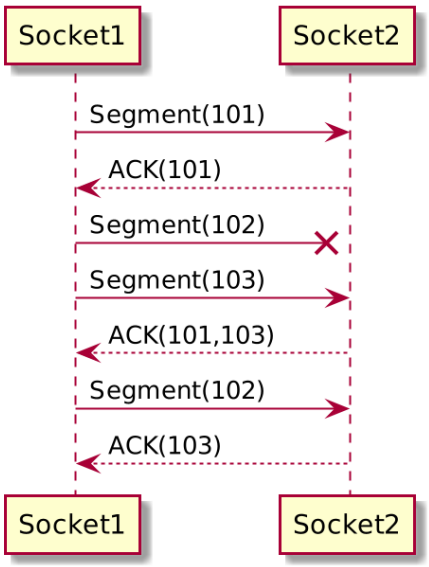

Figure 4: Data segment lost at any point

If a segment is lost in the middle of a transmission, the implemented ARQ protocol will send a request for the missing segment(s) with the next ACK. Since ATP implements selective repeat

ARQ, only the missing segment(s) will be requested and retransmitted while the received out-of-order frames will be buffered for later use. With this variant of ARQ, ACKs will contain a list of received sequence numbers since the packet loss, or the last segment received if no loss was detected.

6.1.4 ACK segment lost

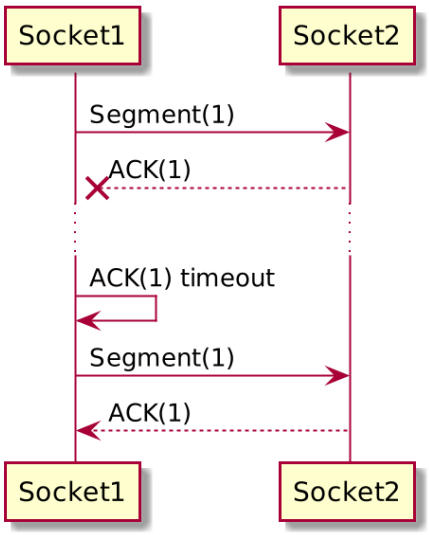

Figure 5: ACK segment lost

In case of a singular ACK segment getting lost, the data segment will eventually timeout and be sent again. The receiver side will ignore the payload and only send the lost ACK again.

6.1.5 Cumulative ACK

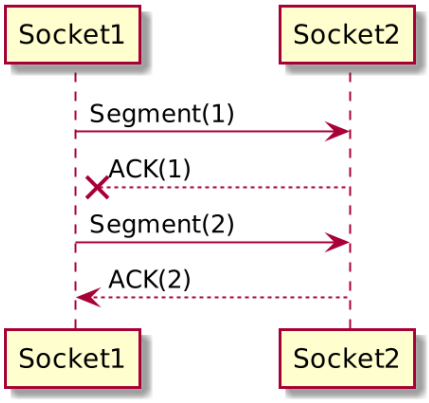

Figure 6: Cumulative ACK

ATP's implementation of ARQ will support cumulative acknowledgement. Thanks to this, any lost ACK will not lead to a timeout and data retransmission as long as later ACKs are not lost. Since the

sender receives ACKs with higher sequence numbers than the segment that wasn't acknowledged, he can infer that that segment must have reached its destination as well.

Part III
Appendix

List of Figures

Glossary

AES-GCM The Advanced Encryption Standard (AES) is a specification for the encryption of data. AES-GCM is a mode of operation for symmetric-key cryptographic.

ARQ Automatic repeat query (ARQ), is an error-control method for data transmission that uses acknowledgements and timeouts to achieve reliable data transmission over an unreliable service.

DHT A distributed hash table (DHT) is a class of a decentralized distributed system that provides a lookup service similar to a hash table.

Diffie-Hellmann key exchange With Diffie-Hellmann key exchange it is possible to securely exchanging cryptographic keys over a public channel.

DTLS Datagram Transport Layer Security (DTLS) is a communications protocol that provides security for datagram-based applications by allowing them to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

Github GitHub is a development platform. You can host and review code, manage projects, and build software.

NAT Is a technique for establishing a direct connection between two clients, where one or both are behind firewalls or behind routers that use network address translation (NAT).

Noise Protocol Framework Noise is a framework for building crypto protocols [5].

TCP TCP provides reliable, ordered, and error-checked delivery of a stream of octets (bytes) between applications running on hosts communicating via an IP network [6].

TomP2P TomP2P is a DHT with additional features, such as storing multiple values for a key [7].

UDP With UDP, applications can send messages to other hosts on an Internet Protocol (IP) network [8].

References

[1] (2019). Hypertext transfer protocol version 3 (http/3), [Online]. Available: `https://quicwg.org/base-drafts/draft-ietf-quic-http.html` (visited on 10/16/2019).

[2] xtaci. (2019). Xtaci/kcp-go: A crypto-secure, production-grade reliable-udp library for go with fec, [Online]. Available: `https://github.com/xtaci/kcp-go` (visited on 10/14/2019).

[3] G. Fairhurst. (2002). Advice to link designers on link Automatic Repeat reQuest (ARQ), [Online]. Available: `https://tools.ietf.org/html/rfc3366#page-2` (visited on 10/12/2019).

[4] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 5th. USA: Addison-Wesley Publishing Company, 2009.

[5] (2019). Noise framework - official homepage, [Online]. Available: `http://noiseprotocol.org/` (visited on 12/16/2019).

[6] J. Postel. (1981). TRANSMISSION CONTROL PROTOCOL, [Online]. Available: `https://tools.ietf.org/html/rfc793` (visited on 10/16/2019).

[7] T. Bocek. (2019), [Online]. Available: `https://github.com/tomp2p` (visited on 10/16/2019).

[8] J. Postel. (1980). User Datagram Protocol, [Online]. Available: `https://tools.ietf.org/html/rfc768` (visited on 10/16/2019).