A SDN Based Method of TCP Connection Handover

Andrej Binder(✉), Tomas Boros, and Ivan Kotuliak

Faculty of Informatics and Information Technologies,
Slovak University of Technology in Bratislava, Ilkovičova 2,
842 16 Bratislava, Slovakia
andrej@binder.sk, tomas.boros92@gmail.com,
ivan.kotuliak@stuba.sk

Abstract. Today, TCP is the go-to protocol for building resilient communication channels on the Internet. Without much overstatement, it can be said that it runs the majority of communication on the planet. Its success only highlights the fact that it also has some drawbacks, of which one of the oldest ones is the inability to hand over running connections between participating hosts. This paper introduces a method that relies on the advantages of Software Defined Networks to overcome this limitation.

Keywords: Software Defined Networks · Network protocols · Transmission control protocol · Telecommunications

1 Introduction

TCP handover is the act of handing over the role of one of the two communicating endpoints to a third endpoint that was initially not a part of the communication.

The reasons for this can be for example:

- Load-balancing
- Traffic path optimization
- A transparent redirection mechanism
- Switchover of network interfaces

The common solution for this problem was to terminate the running connection and re-initiate the connection with a new host. This is a common practice on the interned today [5, 13].

The problems with this approach are:

- Latency caused by additional TCP handshake
- Needs to be implemented on application layer
- Non-transparent
- TCP-windows are reset resulting in sub-optimal performance

One area where this problem is especially apparent is the area of Content Delivery Networks. Most CDN architectures leverage a redirect mechanism to initiate connection between a client and the most appropriate server to serve specific content.

I. Khalil et al. (Eds.): ICT-EurAsia 2015 and CONFENIS 2015, LNCS 9357, pp. 13–19, 2015.
DOI: 10.1007/978-3-319-24315-3_2

Introducing delays in this step results in noticeably slower content playback startups that are even more apparent in the case of CDN Federations where multiple redirects often take place before the client can connect to the server [3, 6, 7, 8].

Our method to address this issue is to make use of Software Defined Network technology. This technology makes it possible to enhance the network with the functionality that not only allows TCP handovers but also makes them controllable by the SDN controller itself [9, 15].

2 Software Defined Networks

The main disadvantage of traditional network technologies is lack of flexibility in implementing new features. Because of requirements related to standardization, testing and the drawbacks of deploying new code in a fully proprietary environment, new features usually take years to be agreed upon. Even then they often face limited success because of the difficulties and risks related to changing something in an environment that was essentially designed to serve a very specific purpose. One example of such technology is multicast that has existed for decades but did not succeed in being globally distributed because of the reasons listed above [1, 10–12, 14].

Software Defined Networks (SDN) present a radically different approach to designing networks that is built from ground up to make implementation of new features and services as easy as possible. It achieves this by splitting the data plane (responsible for forwarding traffic) and the control plane (responsible for higher level decision-making and configuration of the data plane) into two separate entities. Furthermore it also changes the logical placement of these entities. In traditional networks both the control plane and the data plane was confined within a single networking device, making development of complex control plane to control plane communication protocols necessary. In SDNs the data plane stays distributed but the control plane is removed from the physical device and placed into a centralized node responsible for managing all the data planes in the network. This centralized control plane is called a Controller in SDN terminology [2, 9].

The SDN Controller is a fully software-based element that does not have the burden of having to communicate every single decision to any of its peers. This means that new features can be quickly added to the controller and they will be instantly available throughout the whole network that is under its control. The resources available in the data plane under its controls are the Controller's only limiting factor. It does not have to follow a specific protocol that dictates exactly how these resources should be used [4].

3 TCP Handover Method in SDN Networks

Our approach to addressing TCP handover relies on the following features of Software Defined Networks:

- The ability to intercept specific packets and redirect them for processing in the control plane

- The ability to modify the data plane in such a way that it rewrites the destination IP address of a packet according to a rule defined by the control plane
- The fact that a SDN network is limited to a single autonomous system (administered by a single organization), in which the occurrence of triangular routing is not considered a problem as long as its fully controlled

In addition to these requirements at least one of the following features must also be available:

- The ability to synchronize (increment or decrement according to a rule) TCP SEQ and ACK numbers in the data plane
- The ability to synchronize (increment or decrement according to a rule) TCP SEQ and ACK numbers in the host device
- The ability to be able to predict the SEQ number that would be chosen by a host for a new incoming connection (described later)

The initial use case that this method was designed for was the implementation of a transparent redirect mechanism for use in Content Delivery Networks so we will use this environment to describe the method's operating principle. We will later describe how to use the method in any other scenario.

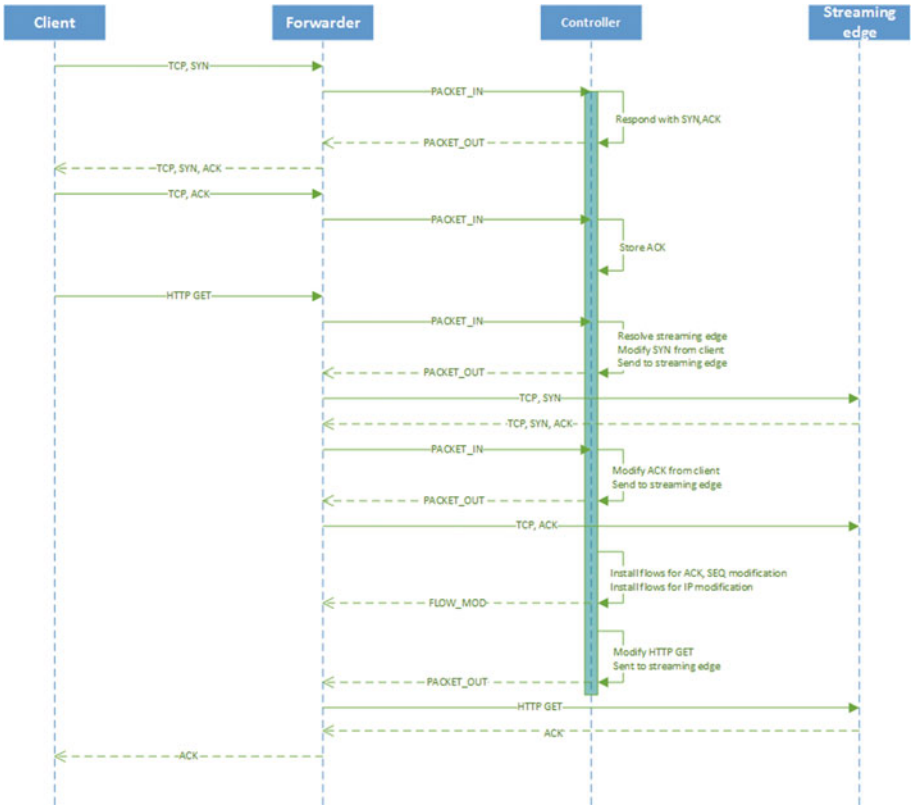
When a client initializes a new TCP connection to a server, it sends a TCP segment encapsulated in IPv4 or IPv6 packet to a destination address that identifies the service to be accessed. In the first TCP segment the client sets the SYN flag, chooses a initial sequence number (SEQ), sets the ACK number to 0, sets the window size and optionally sends some OPTION parameters. The server on the receiving side goes to the SYN_RCVD state and sends back his TCP packet and parameters to the client. Sets the SYN and the ACK flag, choses a sequence number and sets the ACK number to sequence number + 1 of the client. Using this he acknowledges the client to send the next TCP window. The server chooses a window size too and sets some optional parameters in OPTION fields. The client then sends back an ACK message to acknowledge the parameters of the server. At this moment the session goes to ESTABLISHED state. Now the client may request the data (for example in a HTTP GET message).

This happens normally in networks but lets say that the IP address that the client was communicating with was not an IP address directly attached to a specific server but an IP address defined in the network as and address used to identify a specific service. Any TCP packets sent to this IP, that are meant to initiate a TCP connection with a server, will not be delivered directly to a server but redirected to a SDN Controller for further processing instead. The controller would then keep on acting on behalf on the server up to the point when it can decide which actual server would be best to deliver the service. In the context of CDN networks this means up until the point when the Controller is aware of the HTTP URI that the client intends to access. It would then modify the data plane to rewrite the destination IP address of all future packets from the client to the IP address of the chosen server.

This would work perfectly in an UDP-based scenario where packets are considered as separate atomic elements. In TCP environment all communication is treaded in the context of sessions that are kept consistent by communicating the sequence numbers of

packets in each transmission and acknowledging them on the other side. The problem with this approach in context with our method is that we cannot control the initial SEQ number that the client choses or the SEQ number that the final server would chose. This means that, without addressing this issue, the communication would not work because even if the source and destination addresses of the packets were correct, the TCP session would not work because both sides would not be able to agree on which sequence number should follow.

The full method is depicted in the following sequence diagram:



There are two basic ways to address this:

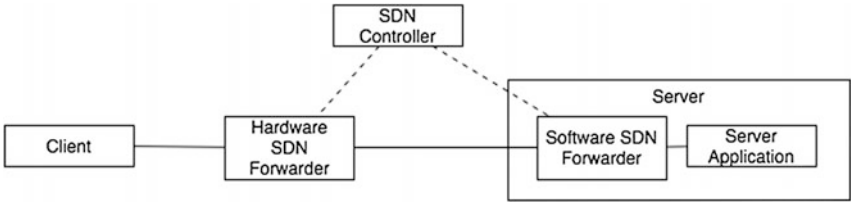
- Be able to synchronize the SEQ and ACK numbers by incrementing or decrementing them in the data plane
- Be able to predict the SEQ number that the server will chose so that the connection from the controller can be started with a SEQ number that would be in sync with the SEQ number that the server would chose right from the start

The first approach has the only disadvantage that the SDN data plane (also called SDN Forwarder) closest to the server would need to have the capability to increment

and decrement the SYN and ACK numbers according to a chosen rule. The fact is that while rewriting of destination IP address is a standard SDN data plane function that is available in basically all SDN Forwarders, the functions of incrementing or decrementing of SYN and ACK numbers are not standard functions. This means that most hardware data plane elements would not be able to perform the operation.

This can be easily addressed in environments where we have the server under control. We simply place a small SDN Forwarder in the operating system of the server and link it to the controller. This small forwarder would be a data plane element only capable of doing the operation of synchronizing the SYN/ACK numbers, an operation that is very easy to implement in the all-software environment of a server.

The following figure depicts this scenario:



The second approach requires the modification of the TCP stack on the server. The SEQ number the TCP stack choses would not be chosen randomly as it is usually done, but it will be chosen according to a hash of the incoming SYN packet. This means that when a SYN packet is sent to such server, the sender has the ability to calculate and predict the initial SEQ number that the server will chose. In order to maintain security, a shared secret (shared between the controller and the server) can also be added to the SYN packet in order to make it harder for a third party to step into the communication.

4 Implementation and Testing

To prove that the approach is fully functional, we have implemented a prototype and tested it with real clients and servers in the environment of CDN networks.

We have created a new version of the Ofsoftswitch13 SDN Forwarder with the additional TCP SEQ and ACK synchronization functions. We did this by adding a new action based on to the SET_FIELD action defined by the OpenFlow 1.3 standard. We called these actions SET_TCP_ACK and SET_TCP_SEQ in order to be able to modify the ACK and SEQ numbers respectively.

For example if we install an action with SET_TCP_SEQ with argument 1000, incoming TCP connection which matches the matching rule will have its Sequence number incremented by 1000 on the outgoing interface. The same thing will happen for the ACK number. Using correctly these actions we will be able to synchronize the TCP sequence and acknowledge numbers for the two separate TCP connections.

In addition to the modified SDN Forwarder we also needed our own SDN Controller that we could easily modify. In the end we chose the Ryu SDN controller. It is an

open-source SDN controller that is freely available, well documented and easy to modify. This controller also fully supports the OpenFlow 1.3 protocol which allowed us to re-use most of the needed functionality. The controller was also modified to be able to track the state of the session in order to get more visibility into what is happening in the network.

These two components allowed us to fully test our method. The testing also shower that in addition to proving that the method actually works, it also has the following benefits:

- Faster session establishment and shorter interruption in comparison with application-level redirect methods
- No need for extra DNS queries
- No need to implement application-level redirect mechanisms
- Fully transparent to the client

5 Conclusion

We designed, implemented and thoroughly tested a new method of TCP connection handover in the environment of SDN networks.

We have created a prototype SDN Forwarder and a prototype SDN Controller that we used to prove the functionality of the method.

Our tests using these prototypes proved that we can achieve faster handover times as compared to traditional application-level redirect methods that require a complete re-establishment of TCP sessions. Furthermore this was all done in a manner that is fully transparent to the client and requires no modification of the server application.

In the end the method proves that SDN technology is a great platform for implementing interesting new functions into the network environment.

Acknowledgements. This work is a result of the Research and Development Operational Program for the projects Support of Center of Excellence for Smart Technologies, Systems and Services, ITMS 26240120005 and for the projects Support of Center of Excellence for Smart Technologies, Systems and Services II, ITMS 26240120029, co-funded by ERDF.

The authors would like to thank Oskar van Deventer and his team at the Dutch Organization for Applied Scientific Research (TNO) for their invaluable help.

References

1. Feamster, N., Rexford, J., Zegura, E.: The road to SDN: an intellectual history of programmable networks, December 2013
2. OPEN NETWORKING FOUNDATION.: OpenFlow Switch Specification: Version 1.3.0 Implemented (Wire Protocol 0x04) (2012)
3. Niven-Jenkins B., Le Faucheur F., Bitar N.: Content Distribution Network Interconnection (CDNI) Problem Statement

4. TEAM, RYU PROJECT. RYU: SDN Framework (Online). Ryu book. http://osrg.github.io/ryu-book/en/Ryubook.pdf
5. Network Working Group. RFC: 2616 - Hypertext Transfer Protocol – HTTP/1.1
6. van der Ziel, S.: CDN interoperability reality check
7. ETSI TS 182 032 CDN Interconnection Architecture
8. Bertrand, G., Le Faucheur, F., Peterson, L.: Content Distribution Network Interconnection (CDNI) Experiments, Internet Engineering Task Force, draft-bertrand-cdni-experiments, 02 February 2012. http://tools.ietf.org/id/draft-bertrand-cdni-experiments-02.txt
9. Kim, M.-K., Kim, H.-J., Chang, D., Kwon, T.: CDNI Request Routing with SDN, Internet Engineering Task Force, draft-shin-cdni-request- routing-sdn-00, July 2012. http://tools.ietf.org/id/draft-shin-cdni-request-routing-sdn-00.txt
10. Kokku, R., Rajamony, R., Alvisi, L., Vin, H.: Half-Pipe Anchoring: An Efficient Mechanism for TCP Connection Handoff
11. Bonaventure, O., Handley, M., Raiciu, C.: An Overview of Multipath TCP
12. Beda, E., Ventura, N.: Socketless TCP - an end to end handover solution (2014)
13. IETF RFC 793.: Transmission control protocol, September 1981. http://www.ietf.org/rfc/rfc793.txt
14. Kozemčák, A., Kováčik, T.: Different network traffic measurement techniques - possibilities and results. In: Proceedings ELMAR-2012: 54th Symposium ELMAR-2012, pp. 93–96. Society Electronics in Marine, Zadar, 12–14 September 2012. ISBN: 978-953-7044-13-8
15. Halagan, T., Kováčik, T.: Modification of TCP SYN flood (DoS) attack detection algorithm. In: Numerical Modelling and Simulation : International Interdisciplinary PhD Workshop IIPhDW, Tatranske Matliare, Slovak republic, 20–22 May 2014. 1. vyd. Warsaw : Elektrotechnical institute, 2014, [4] s. ISBN: 978-83-61956-29-7