Transparent Redirection in Content Delivery Networks using Software Defined Networking

Tomáš Boros[*]
Institute of Computer Engineering and Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 2, 842 16 Bratislava, Slovakia
tomas.boros@stuba.sk

Abstract

The Internet is the technology of choice for mass distribution of information today. Content Delivery Networks, as the method used to deliver majority of the Internet's content, are becoming the most important piece of making this kind of communication sustainable. They do however impose some limitations as opposed to older legacy approaches. Content Delivery Networks heavily rely on redirections, which directs user requests to the closest surrogate server to lower the delay for content delivery and distribution on a massive scale. Redirections have not received a significant amount of attention in the recent year. This paper will show a TCP session handoff can be achieved using Software Defined Networking, which can be used a transparent redirection mechanism in Content Delivery Networks. The paper presents a working prototype including the achieved results with comparison with existing legacy approaches.

Categories and Subject Descriptors

C.2.0 [**Networks**]: General; C.2.1 [**Networks**]: Network Architecture and Design; C.2.3 [**Networks**]: Network Operations; C.2.6 [**Networks**]: Internetworking

Keywords

Software Defined Networking, Network Protocols, Transmission Control Protocol, Content Delivery Networks, TCP Session handoff, Session handover

1. Introduction

Content Delivery Networks (CDN) are systems for efficient delivery of digital objects (e.g. files with multimedia content such as video on demand or other file types) and multimedia streams (e.g. live television streams) over IP networks to many end points and viewers. Typically, a CDN consists of one or more servers that deliver the digital objects and/or streams, and a management/control system. The management/control system takes care of content distribution, request routing, reporting, metadata and other aspects that make the system work. [8]

There are two main types of CDNs with global coverage today. The first type is a single CDN operated by a single entity that is managing all the infrastructure globally. Some examples of such CDNs are Akamai, Google Cache, Limelight, Level 3 and others. The second kind is represented by a group of smaller CDNs with limited coverage that, when combined, create a single service with global coverage. This kind of CDN is called a CDN Federation [4].

One of the main drawbacks is delay introduced in CDN. In this article, we propose a novel approach to session redirection using an SDN-based TCP session handoff. Using this approach, the delay becomes similar to DNS based redirection while having the accuracy of application layer redirection. The redirection occurs fully transparently to all the components. Furthermore, the proposed solution does not require any additional changes to the endpoints.

The rest of the paper is organised as follows: Section II gives insight into the state of the art. Section III is the core of our proposal using a TCP handoff mechanism. Section IV introduces the used testbed which was used to evaluate the prototype. Section V presents the achieved results and the final conclusions are given in Section VI.

2. State of the Art

2.1 CDN Redirection Methods

Redirection is the crucial part of the content delivery. A proper redirection mechanism ensures that the user requests are redirected to the closest optimal surrogate server in the CDN network. Barbir et al. in RFC3568 lists and compares the known CDN request-routing mechanisms [3]. The RFC lists the following types of redirect mechanism:

- DNS based Request-Routing mechanism
- Application-Layer Request-Routing mechanism
- Transport-Layer Request-Routing mechanism

[*]Recommended by thesis supervisor: Prof. Ivan Kotuliak
To be defended at Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava on [ToDo].

Each mechanism has its advantages and drawbacks. DNS based redirection works by redirecting the client to the surrogate server during the DNS resolve. In such case the Request Router in the CDN works as a DNS server and dynamically returns an A or AAAA record based on the client's IP address or the client's DNS server's IP address if the DNS translation occurs in a recursive way. The DNS response points to a surrogate server, which delivers the content to the client. Such redirection is fast, however does not always returns the optimal surrogate server. [17]

Application layer mechanism occurs on the application layer of the TCP/IP model. In such case the request router works as a surrogate server, however does not return any content but only generates redirection messages to a surrogate server which caches and delivers data to endpoints. Each content distribution protocol which supports redirection may be used in CDN e.g. HTTP, RTSP, RTMP, DASH etc. Such redirection method is more accurate as application layer parameters may be analyzed during the redirection such as User Agent, URI during HTTP or available codecs in SDP during RTSP redirections. As the redirection message must be processed and often an additional domain name resolve must occur if the redirection message contains a domain instead of an IP address, this redirect method generates delays on session initiation. To speed up the redirection often a URL rewriting mechanism is used, which modifies the contents' URL in HTML to point directly to the surrogate server [3].

Transport-Layer Request-Routing is not a commonly used redirection method as it requires a service awareness and support from the network too. Transport-Layer request routers use information in the transport-layer headers to determine which CDN surrogate server should serve the client. The request router examines the IP address and port numbers in the TCP SYN and forwards to an appropriate CDN server. The target CDN server establishes the connection and proceeds to serve the requested content. Forward traffic from the client goes to the request router which is then forwarded to the CDN server but the response from the CDN server travels on the direct path to the client in a triangular routing way. This can be achieved using IPv6 too. IPv6 supports mobility which eliminates the triangular routing, however the majority of the clients are still using IPv4 [2].

2.2 Software Defined Networks

Traditional network technologies lack flexibility in implementing new features. Implementing new features and protocols may take years due to required standardizations, testing and doployment of the new code in a fully proprietary environment [16]. Software Defined Networking (SDN) presents a radically different approach. The key advantage is that the control plane is separated from the data plane. In traditional networks both the control plane and the data plane were confined within a single networking device, making development of complex control plane to data 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. The data plane consists of a network of SDN forwarders which are not capable of making decisions. The data plane forwards and processes packets based on match rules and actions installed in their flow tables. If a does not match any rule, the forwarder might send the packet to the control plane via the Southbound API for further processing [20]. The control plane consists of a controller, a separated node which controls the data plane from a centralized view. It communicates with the forwarders in the data plane via the Southbound API and feeds the forwarders with instructions how to handle their incoming communications. One of the common protocols used in this API is OpenFlow. This centralized control plane is called as SDN Controller in SDN terminology. 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 SDN Controller can control multiple SDN forwarders in the network. 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. New features can be easily implemented and added to the network via the Controller's Northbound API. [5]

3. Transparent CDN Redirection Using SDN

Transport-Layer Request-Routing is transparent to the client, however it does not take into consideration the application layer parameters during redirection. To achieve higher accuracy, we must intercept packets on the data plane. In SDN world it is very easy to capture and analyze certain type of packets. Upon successful analysis, we can easily handover the session to the appropriate surrogate server. Such redirection is very easy for UDP based communication as UDP is stateless, however TCP is more often used as HTTP is encapsulated in TCP transport segments [1] [19].

TCP is a stateful transport protocol and the sessions are identified by a 4-tuple of source/destination IP addresses and ports. Data sent during a session must be acknowledged by the receiving side to ensure reliable data transfer. Each segment of data is labeled with a sequence number which is then acknowledged by the acknowledge number on responses. Sequence number is chosen dynamically on session initialization so it is not possible to hand over sessions easily. There were various attempts to hand off an existing live TCP sessions. Wichtlhuber et al. used the `TCP_REPAIR` flag in Linux kernels to migrate the socket form the terminating BRAS to a surrogate server, however such approach works only for Digital Subscriber Lines (DSL) and requires modification on the surrogate server to be able to accept the migrated session [21]. Guerney Hunt, Erich Nahum and John Tracey in IBM T.J. Watson Research Center already came up with an idea of handoff an existing TCP session from a load balancer to a content server in 1997 [15]. Authors in this report introduce the handoff mechanism of TCP sessions from the load balancer, where the initial TCP session from the client is established. After establishment, the clients send the HTTP request, which is examined and inspected by the load balancer, and based on the requested content, the TCP session is handed off to a content server, which has the requested content cached. The initial TCP session parameters between the client and the load balancer are passed to the cache server (Windows sizes, various TCP options, Sequence Numbers) re-using the same TCP session parameters between the load balancer and the cache server. Once the session is handed off, the communication flows directly between the client and the cache server, while the IP address of the cache server is rewritten, to mimic the load balancer. It is not clear from the report, whether a working prototype was implemented or not. Authors also mention a different approach using T/TCP (Transactional TCP) where the initial HTTP request can be sent already in the 3 way handshake phase [6] [7]. This protocol is faster than TCP and delivery reliability is comparable to that of TCP. T/TCP suffers from several major security problems as described by Charles Hannum [11] [12]. It has not gained widespread popularity and the protocol was moved to historic status in May 2011 by RFC 6247. Authors also mention in the report a commercial product by Resonate Inc, which adopts the same mechanism, which they call âĂŸconnection hop', however, the product does not seem to be existing anymore [9]. Yutaro Hayakawa, Michio Honda, Lars Eggert and Douglas Santry in their paper [13] present Prism proxy for fast load balancing technique, where the TCP sessions established with the load balancer, upon reception of the request, are handed-off to a chosen backend server from the pool. The solution is designed for datacenter scale load balancing; however, it could be used for CDNs as a transparent redirection engine over greater areas. The Prism proxy uses the same TCP_REPAIR [10] flag to pass the sessions between the servers. The solution uses HTTP/1.1 and is capable of reusing existing TCP sessions for subsequent requests. The authors used mSwitch, which is a kernel software switch that can forward packets at a rate over 10 Mpps on a single CPU core [14]. To perform a transparent redirection the CDN must be able to perform a TCP session handoff, otherwise, the redirection becomes visible to the end user. All the mentioned related work above points out the challenges associated with the handoff procedure. In our proposal we do not hand off an existing TCP session from the Request Router to the Surrogate server, but instead we synchronize two different TCP sessions with each other by rewriting TCP and IP header fields including the acknowledgement and sequence numbers in the TCP header. The full session handover is depicted in the sequence diagram in Figure 1. Figure 1 is divided into 4 sections.

To make this solution work we must synchronize the sequence and acknowledgement numbers in the TCP headers too. Such modification could be implemented on a data plane as an experimenter action. The action must be able to increment these numbers to map the other TCP session's values. Once the sequence/acknowledge number reaches the maximum value of a 32bit number, it overflows and starts from 0. Upon a successful installation of actions to the data plane, the client consumes the response message from the surrogate server, which was originally generated for the request router (1st TCP session), while the client âĂIJthinksâĂİ the response is coming directly from the request router (2nd TCP session). In the end, to finish the handover procedure, the SDN controller must create reset (RST) or FIN messages to mitigate both sessions which were established with the request router. In order to be able to perform the handoff we have to keep track of the existing TCP sessions in the network. This can be achieved by intercepting the initial TCP handshakes with the request router and the surrogate servers. The request router prepares a pool of established TCP sessions towards the surrogate servers, which are reused for the handoff. Once the client establishes a TCP session with the request router a valid HTTP

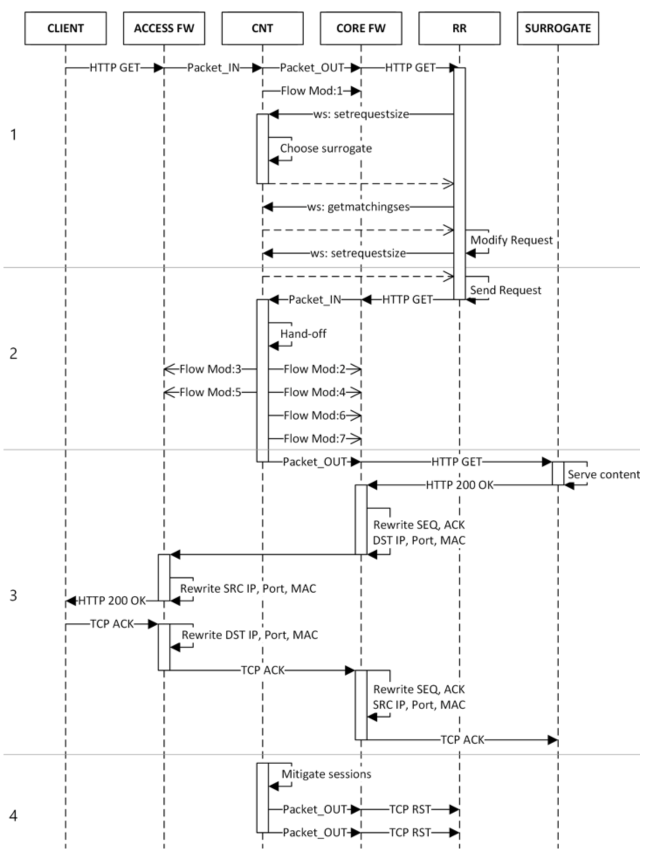

Figure 1: TCP session handover flow diagram.

request message is expected. From this point the handoff procedure begins. The solution is designed to work with the standardized HTTP 1.1. In the 1st section of figure 1 the clients send an HTTP GET request including the URL and additional request headers. This request is sent to the controller (CNT) and then forwarded to the request router. Once the full request is sent to the request router the controller blocks the communication from the request router towards the client (Flow Mod:1). The request router parses the request and reports the request size to the controller over over the Northbound API via the setrequestsize message. This confirms the controller, that a full request was received. The request router requests the surrogate server to which the session must be handed off via the getmatchingsess message. Based on the returned information the request router modifies the request by updating the Host header in the HTTP request and reports the request's size to the controller via the setrequestsize message again. In the section 2 of figure 6 the request router sends out the request towards the surrogate server via an already pre-established TCP session (this session was chosen by the controller). At this point the controller has all the required information to perform the handoff. The controller installs flow mods to the core forwarder, where the surrogate is connected and to the access forwarder where the client is connected. The following flow mods are installed:

- Flow mod 2: rewrites destination IP, port and MAC address for the communication from the surrogate server towards the client, where the IP, port and MAC address is rewritten to mimic the request router. Sequence number and Acknowledge number is incremented to synchronize with the TCP session established between the client and request router

- Flow mod 3: rewrites the source IP, port and MAC address to match the parameters of the client. The packet sent out from the surrogate server is originally destined to the request router
- Flow mod 4: rewrites source IP, port and MAC address to match the parameters of the request router for the communication from the client towards the surrogate server. Sequence and acknowledge numbers are incremented to synchronize with the TCP session between the request router and the surrogate server.
- Flow mod 5: rewrites the destination IP, port and MAC address to match the parameters of the surrogate server as the outgoing packet from the client is destined to the request router.
- Flow mod 6 and 7 blocks any further communication from the request router towards the client or surrogate server over the two established TCP sessions.

Intermediary forwarders on the paths have simple match and output actions to forward the packets to the desired destination. In section 3 of figure 6 the modified request is sent out from the controller to the surrogate server, which parses the request and sends a response upon a valid request. The response and acknowledgement packets are forwarded through the network and the parameters are rewritten on the core and access forwarders based on the flow mods from section 2. The original TCP sessions established with the Request Router are mitigated by the controller by sending out TCP RST packets. Once the sessions are mitigated, the request router establishes a new TCP session towards the surrogate server to keep the pool of available TCP sessions alive. The following formulas are used to calculates the differences between the two TCP sessions:

$$Sinc_cs = ((2^{32}) + (Srs - Scr) + (Rrs - Rcr))mod(2^{32})(1)$$

$$Ainc_sc = ((2^{32}) - Sinc_cs)mod(2^{32})(2)$$

$$Sinc_sc = ((2^{32}) + (Src - Ssr))mod(2^{32})(3)$$

$$Ainc_cs = ((2^{32}) - Sinc_sc)mod(2^{32})(4)$$

Where:

- Srs: Initial sequence number from request router to the surrogate server
- Scr: Initial sequence number from client to the request router
- Rrs: Request size from the request router to the surrogate server
- Rcr: Request size from client to the request router
- Src: Initial sequence number from the request router to the client
- Ssr: Initial sequence number from the surrogate server to the request router

As the sequence and acknowledge numbers are 4 byte numbers, modulo 32 is applied to prevent overflow of this number. SeqCS (1) is applied in direction from client to the surrogate server as `inc_seq`(SeqCS) on the core forwarder (Flow mod 4). AckSC (2) is applied in direction from the surrogate server towards client as `inc_ack` (AckSC) on the core forwarder (Flow mod 2). SeqSC (3) is applied in direction from surrogate to client as `inc_seq` (SeqSC) (Flow mod 2) and AckCS (4) is applied in direction from client to surrogate as `inc_ack` (AckCS) on the forwarder where the surrogate server is attached (Flow mod 4). Action `inc_seq` and `inc_ack` are interpreted as datapath actions on the forwarders, which increment sequence and acknowledge numbers respectively.

4. Testbed

To evaluate the proposed solution we implemented a forwarding element, which is capable of incrementing the sequence and acknowledge numbers on the data path. We have chosen OpenVswitch for this purpose. The forwarder consists of two components. A fully userspace implementation of the forwarding datapath called ovs-switchd and a datapath kernel module written specially for the host operating system achieving high performance [18]. In addition to the modified SDN Forwarder we also needed our own SDN Controller that we could easily modify. We chose SDN Controller RYU, which is open-source, well documented and easy to modify. The controller right now support OpenFlow protocol up to 1.5, however as the data plane runs on version 1.3, we had to fall back to this version. The controller is capable of serving a single L2 network and the surrogate servers for the client requests are chosen based on the hop count. We implemented our custom Request Router, which was connected to the network to be able to terminate TCP sessions. The request router communicates with the SDN controller over the Northbound API during the redirection procedure. The testbed allowed us to test the implemented solution. We decided to use a custom build open source CDN solution which is based on Nginx. The implemented prototype allowed us to test the TCP session handover in an SDN network to enable Transparent CDN Redirection. As a result, we have achieved a fast transparent redirection method, which did not expose the physical topology (IP addresses) towards the clients. Some TCP Options (Timestamps, WSCALE, SACK) had to be turned off to make this kind of TCP session handoff work.

5. Evaluation

As next we made delay and bandwidth measurements on a network with 2 hops between the client and the surrogate server and we compared the proposed transparent redirection with the 302 redirection side by side. The topology used is shown in figure 2.

As already mentioned, the transparent redirection requires to be sent over the SDN controller, so the TCP session handshake traverses to the control node via the Packet IN and Packet OUT messages. This add some initial delay to the 3-way handshake. The time required to establish a TCP connection with the request router is shown in figure 3.

Once the TCP session is established, the request router expects a HTTP GET message from the client. Upon a valid request the client is redirected using a 302-redirection method toward the surrogate server, from where

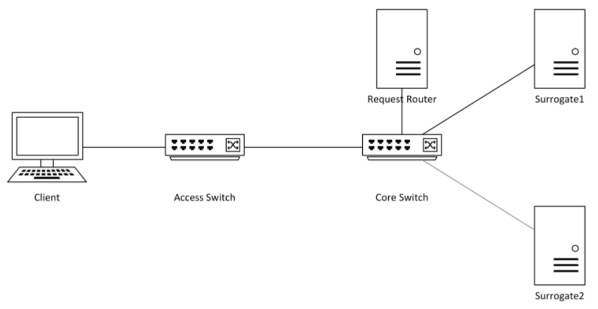

Figure 2: Topology for measuring overall redirection delays.

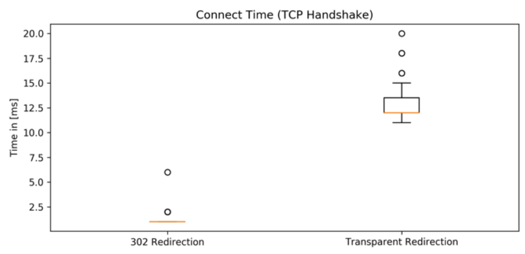

Figure 3: 3-way handshake delay.

the content is served. Using the transparent redirection, a TCP session handover occurs to the surrogate server. Figure 4 shows the time required to process the redirection methods. The time between the sent HTTP GET message and receiving the first data bytes is measured.

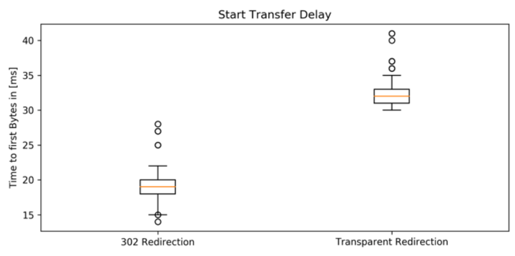

Figure 4: Time required to process redirection.

To give some insight into the performance of the data plane, we did measurements on a large file (1GB). This size of the file is enough to reach high performance on the data plane due to the increasing window size. Figure 5 shows the whole time required to download the 1GB file from the surrogate server including TCP session establishment and redirection.

After each test we reported the mean bandwidth reached during the file transfer, which we put on a boxplot to evaluate the performance of the data plane itself. Figure 6 shows the average bandwidth reached during the file transfers. We have to note, when 302 redirection is used the forwarding elements only match the packets and forward them to the next hop based on output action in the flow table. No any other modifications are done on the packets. When transparent redirection is used, due to the TCP session synchronization, source IP, destination IP, source port and destination port are rewritten and sequence and acknowledge numbers are incremented.

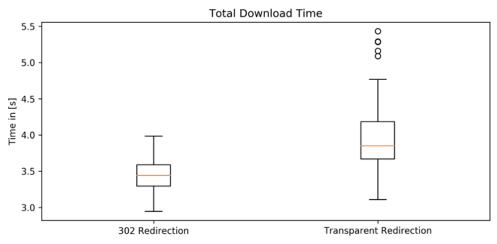

Figure 5: Total time required to download file.

As we made our tests on a L2 network, source and destination hardware (MAC) addresses were rewritten too. This comes to a fact, that transparent redirection does 8 modifications on the packets, while the 302 redirection does not do any changes. Any change applied to a TCP or IPv4 Header also triggers a recalculation of the CRC checksum. Based on the achieved bandwidth (280 Mbit/s compared to 320 Mbit/s), we can state, that the modifications made on the Open vSwitch forwarder did not degraded the overall performance.

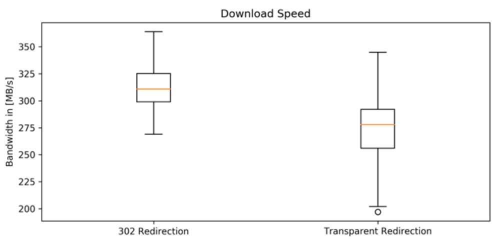

Figure 6: Average bandwidth reached.

6. Conclusions

The paper proposes a TCP session handoff mechanism as a transparent redirection which synchronizes two existing TCP connections with each other. The main contribution of this project a prototype, which allows to hand off existing TCP sessions from one endpoint to the other without any further modifications to the endpoints. The transparent redirection ensures that the TCP session is handed off to a CDN surrogate from the request router, while the procedure stays unnoticeable to the endpoints. The redirection decision is based on the network topology and the location of the surrogates and the users. The results acquired show that current redirection methods (302 Redirection) are faster than the proposed solution, however, this performance could be enhanced by using a more mature and faster controller. Currently, the controller represents a single point of failure, does not scales and represents our performance bottleneck in the proposed solution. There were several issues found, which should be solved in the future. One of them is the issue of pre-establishing TCP sessions form the request router to the surrogate servers with various WSCALE options to be able to make exact matches when we synchronize the TCP sessions. The incrementation of the acknowledgement numbers in the selective acknowledgement TCP Option should be implemented on forwarding elements too. Currently, these TCP Options including TCP Timestamps are turned off, which could degrade the performance in challenging environments. Furthermore, in the future, the possibility of using secured TLS based TCP sessions should be elaborated.

References

[1] Cisco visual networking index: Forecast and methodology, 2016-2021, white paper.

[2] A. Acharya and A. Shaikh. Using mobility support for request-routing in ipv6 cdns. In *Proceedings of Proc. 7th International Workshop on Web Content Caching and Distribution 2012*, 2012.

[3] A. Barbir, B. Cain, R. Nair, and O. Spatscheck. *Known Content Network (CN) Request-Routing Mechanisms*. RFC Editor, United States, July 2003.

[4] A. Binder and I. Kotuliak. Content delivery network interconnect: Practical experience. In *2013 IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 29–33, Stara Lesna, Oct 2013.

[5] A. Bondkovskii, J. Keeney, S. van der Meer, and S. Weber. Qualitative comparison of open-source sdn controllers. In *in NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, pages 889–894, 2016.

[6] R. T. Braden. Rfc 1379 - extending tcp for transactions – concepts, 1992.

[7] R. T. Braden. Rfc 1644 - t/tcp–tcp extensions for transactions functional specification, 1994.

[8] R. Buyya, M. Pathan, and A. Vakali. *Content Delivery Networks*. Number 5. November 2010.

[9] M. Cooney. Getting corporate intranets under control. *Network World*, 13(51):17–17, 1996.

[10] J. Corbet. Tcp connection repair. https://lwn.net/Articles/495304/, 2012. Accessed: 14-Apr-2019.

[11] C. Hannum. Security problems associated with t/tcp, 1996.

[12] C. Hannum. T/tcp vulnerabilities. *Phrack Magazine*, 8(53), 1998.

[13] Y. Hayakawa, L. Eggert, M. Honda, and D. Santry. Prism. In *in Proceedings of the 2017 Symposium on Cloud Computing - SoCC '17*, pages 181–188, 2017.

[14] M. Honda, F. Huici, G. Lettieri, and L. Rizzo. mswitch. In *in Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research - SOSR '15*, pages 1–13, 2015.

[15] G. Hunt, E. Nahum, and J. Tracey. Enabling content-based load distribution for scalable services. 09 2002.

[16] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, Jan 2015.

[17] J. Otto, M. Sánchez, J. Rula, and F. Bustamante. Content delivery and the natural evolution of dns: Remote dns trends, performance issues and alternative solutions. In *IMC 2012 - Proceedings of the ACM Internet Measurement Conference*, Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, pages 523–536, 12 2012.

[18] B. Pfaff et al. The design and implementation of open vswitch. In *NSDI'15 Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, pages 117–130, 2015.

[19] J. Postel et al. Rfc 793: Transmission control protocol, 1981.

[20] V. Šulák, P. Helebrandt, and I. Kotuliak. Performance analysis of openflow forwarders based on routing granularity in openflow 1.0 and 1.3. In *2016 19th Conference of Open Innovations Association (FRUCT)*, pages 236–241, Nov 2016.

[21] M. Wichtlhuber, R. Reinecke, and D. Hausheer. An sdn-based cdn/isp collaboration architecture for managing high-volume flows. *IEEE Transactions on Network and Service Management*, 12(1):48–60, March 2015.

Selected Papers by the Author

A. Binder, T. Boros, I. Kotuliak. . A SDN based method of TCP connection handover. In *Information and communication technology : Third IFIP TC 5/8 International Conference (ICT-EurAsia 2015), and 9th IFIP WG 8.9 Working Conference (CONFENIS 2015), Held as Part of WCC 2015*, pages 13–19, Daejeon, Korea, 2015. Springer.

T. Boros, P. Zuraniewski, R. Hindriks, N. van Adrichem, E. Thomas, L. D'Acunto. Enabling superior and controllable video streaming QoE with 5G network orchestration. In *2019 22nd International Conference on Innovation in Clouds, Internet and Networks (ICIN 2019)*, pages 124–129, Paris, France, 2019. IEEE.

T. Boros, I. Kotuliak. SDN-based transparent redirection for content delivery networks. In *TSP 2019 : 42nd International conference on telecommunications and signal processing*, pages 373–377, Budapest, Hungary, 2019. IEEE.