Taming the Wild: A Scalable Anycast-Based CDN Architecture (T-SAC)

Qiang Fu, Bradley Rutter, Hao Li, *Member, IEEE*, Peng Zhang, Chengchen Hu, Tian Pan, *Member, IEEE*, Zhangqin Huang, and Yibin Hou

***Abstract*—The prohibitive cost of deploying a sophisticated DNS-based CDN makes anycast-based CDN an attractive alternative for new or small CDN operators. In anycast-based CDNs, user requests are naturally routed to the "closest" server determined by Internet routing. For the operators, however, this comes at a cost—*loss of control*—how the traffic is routed is entirely at the mercy of BGP routing. The "closest" server may be overloaded, or simply not the best choice. This "loss of control" undermines the *scalability* of anycast-based CDN architectures. To have control over how traffic is routed, existing work either requires adding a large amount of complexity to the system (high Capex/Opex) or is unable to achieve precise and fine-grained control. This paper proposes T-SAC, a scalable anycast-based CDN architecture that capitalizes on the programmability and flexibility of SDN/NFV, enabling fine-grained traffic redirection among CDN servers. T-SAC achieves precise control by leveraging a load-based redirection algorithm and a single 1-bit no-redirect flag. We implement T-SAC in the real system and evaluate its performance from various aspects using DASH and web applications. The results show that T-SAC is capable of redirecting the right amount of traffic at the right time to the right servers, making the system highly scalable.**

***Index Terms*—CDN, NFV, scalability, SDN.**

I. Introduction

IT IS estimated that by 2020 82% of Internet traffic will be made up of video traffic, and 73% of the video traffic will be handled by Content Delivery Networks (CDNs) [1]. The prohibitive cost to deploy sophisticated DNS-based CDNs as Akamai does make anycast-based CDNs an attractive alternative for new or small CDN operators. CloudFlare, CacheFly, Edgecast, Amazon AWS, Microsoft Azure and Google Cloud CDN are some of the examples. In anycast-based CDNs, all the cache servers share the same anycast IP address. User traffic is naturally routed to the "closest" server determined by BGP routing [2], [3]. It has been shown that anycast-based CDNs performs better than the traditional DNS-based CDNs [4], even with 20% of these users connecting to sub-optimal servers [5]. However, for the operators this simplicity of anycast comes at a cost—*loss of control*—how the traffic is routed is entirely controlled by BGP. It is difficult to redirect traffic away from the server determined by BGP even if it is overloaded. When a server is overloaded, often it may choose to withdraw its BGP route. However, this will just break all existing TCP sessions and move them to the next "closest" server, making it at the risk of being overloaded. This "loss of control" is a serious challenge to the design of scalable anycast-based CDN architectures.

To have control on how the traffic is routed, the current work either requires adding a large amount of complexity to the system (high Capex/Opex) [6]–[8] and/ or is unable to achieve precise and fine-grained control [4], [6], [9]. To overcome these limitations, we propose T-SAC (Taming the wild: a Scalable Anycast-based Cdn architecture), which takes advantage of Network Function Virtualization (NFV) [10], Software Defined Networking (SDN) [11] and the combination of anycast/unicast for traffic redirection.

OpenFlow-based SDN separates the data plane from the control plane, which is a centralized controller. The controller has a global view of the network and sets up the forwarding rules in the data plane. The centralized controller makes the network highly programmable. NFV, on the other hand, emphasizes the migration of network functions from dedicated hardware to virtual machines running on shared high-volume/ capacity commodity hardware.

We make use of the anycast and unicast IP addresses of a server. When an OpenFlow switch receives a user request destined to the anycast address of a CDN, it may forward the request to an NFV Redirection Node, where the anycast address is translated into a unicast address of a cache server. This server could be the one determined by BGP or a more suitable one identified by the system. The request is then received and processed by the server, which in turn returns the requested content to the user with its anycast address.

Manuscript received March 16, 2018; revised August 11, 2018; accepted August 20, 2018. Date of publication September 19, 2018; date of current version December 11, 2018. The work of Q. Fu and B. Rutter was supported by InternetNZ. The work of H. Li was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61702407 and in part by the Fundamental Research Funds for the Central Universities. The work of P. Zhang was supported in part by NSFC under Grant 61772412 and in part by the K. C. Wong Education Foundation. The work of T. Pan was supported by NSFC under Grant 61702049. *(Corresponding author: Qiang Fu.)*

Q. Fu is with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: qiang.fu@ecs.vuw.ac.nz).

B. Rutter is with Datacom, Wellington 6011, New Zealand (e-mail: bradley.rutter@datacom.co.nz).

H. Li and P. Zhang are with the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: hao.li@xjtu.edu.cn; p-zhang@xjtu.edu.cn).

C. Hu is with Xilinx Labs Asia Pacific, Singapore 486040, and also with the Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: chengchenhu@xjtu.edu.cn).

T. Pan is with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China (e-mail: pan@bupt.edu.cn).

Z. Huang and Y. Hou are with the School of Software Engineering, Beijing University of Technology, Beijing 100124, China (e-mail: zhuang@bjut.edu.cn; ybhou@bjut.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2018.2871323

In T-SAC, the SDN controller may add a forwarding rule to the OpenFlow switch, which then forwards anycast packets to an NFV Redirection Node. The NFV node performs the redirection functionality, determining which CDN server is going to handle the packets. In this way, we are able to move the redirection functionality as well as associated flow entries from the switch (data plane) to the NFV node. As a result, the switch remains simple and scalable only looking after the packet forwarding process, and the valuable TCAM memory is preserved for this purpose (see Section III-A2).

The use of SDN and NFV is essential to the realization of T-SAC. By manipulating the OpenFlow forwarding rules on the fly, the operator can exercise control on whether the traffic is handled by the native anycast or the NFV node, as well as what traffic is handled by which NFV node. The nature of NFV makes it convenient to spin up or shut down an NFV Redirection Node, which can be potentially placed anywhere in the network at anytime, even collocated with the switches. These features make the scalability of the NFV node controllable and allow incremental deployment of the nodes.

Now ISP/CDN operators have control on which server a user is connecting to. From there we design a set of mechanisms including a load-based redirection algorithm and a no-redirect flag to enable precise and fine-grained control on how traffic is routed for load management.

With these mechanisms, the traffic is redirected in a proactive, dynamic, smooth, flexible and non-interruptive manner. The redirection starts proactively when the server is at the risk of being overloaded. Only the right amount of traffic is redirected to maintain the performance of the current server without unnecessarily offloading to other servers. Moreover, the traffic can be potentially redirected to any servers that are suitable to handle additional traffic. Only new flows are redirected. The existing flows stay with the current server without being interrupted. T-SAC is implemented in real systems. Its performance is evaluated through emulation with an extensive set of aspects using DASH and web applications. It shows that the proposed CDN architecture can redirect the right amount of traffic at the right time to the servers that are able to handle the traffic, without breaking existing TCP sessions. We demonstrate that T-SAC is simple yet effective and scalable, suitable for practical deployment. While T-SAC is designed to route traffic within a CDN, it can be extended to route traffic between CDNs in the context of Meta-CDN [12].

The key contributions in this paper are as follows:

- We propose a scalable anycast-based CDN architecture called T-SAC that capitalizes on the programmability of SDN/NFV and the flexibility of virtualization, and takes advantage of ISP/CDN collaboration, giving ISP/CDN operators the control on where the traffic is directed without interrupting existing TCP sessions. For the applications that support concurrent TCP sessions, T-SAC allows the retrieval of content from multiple cache servers, with the concurrent TCP sessions connecting to different servers.
- We design a method for proactive, dynamic and smooth redirection that can handle nicely both sudden bursts and slow movement of traffic, ensuring that the right amount of traffic is redirected at the right time.
- We design a mechanism using a one-bit no-redirect flag that enables flexible redirection, potentially allowing redirection to any cache servers in the CDN.
- We propose a design that makes SDN/NFV scalability issues controllable and allows the incremental deployment and flexible placement of the NFV redirection nodes, which can coexist with native anycasting.
- We prototype T-SAC using Ryu OpenFlow controller framework [13] and Click [14] in Mininet [15] on AWS (Amazon Web Service) [16], and evaluate it with an extensive set of scenarios including the use of node.js [17], Wget [18] and DASH applications [19].

The rest of this paper is organized as follows. Section II analyzes the challenges of cache server selection, which leads to the design of our CDN architecture. Section III describes our architecture and its traffic redirection capability. Section IV covers system level implementation issues. The evaluation of the architecture is presented in Section V. Section VI discusses some key issues related to our architecture. Section VII looks at related work. Finally, we conclude our paper in Section VIII.

II. Server Selection in CDN

CDNs can route a user to a cache server primarily in two ways.

A. DNS-Based CDN

In DNS-based CDN, a client requests content from a CDN via a hostname that belongs to the CDN. The Domain Name System (DNS) is responsible for translating this human-friendly hostname into the IP address of a cache server. DNS is a hierarchical system. The client sends its DNS request to its local recursive DNS resolver (LDNS), which is typically configured by the client's ISP. The LDNS is responsible for identifying and forwarding the DNS request to the authoritative DNS resolver of the CDN, which then discovers and returns the IP address of a suitable cache server (translation). The LDNS receives this translation and returns the IP address of the cache server to the client. The client then connects to the cache server and fetches the requested content. In the meantime, the LDNS caches the translation for the period of the time-to-live (TTL) associated with the translation. In addition, the translation may also be cached in OS or by applications such as web browsers, where the TTL value may be modified [20]. Other clients behind the LDNS requesting content via the same hostname will get the translation from the cache during the TTL.

The decision on which IP address will be returned by the authoritative DNS server is performance based. Often the IP address of the cache server that is geographically close to the LDNS will be returned. Or, advanced techniques may be used to measure the performance of the cache servers and identify the best possible server to serve the client. However, there are a number of issues that are difficult to address with DNS-based CDNs.

1. *Client mis-location.* The cache server associated with the returned IP address may be close to the LDNS. There is no guarantee that the cache server is close to the

client [21], [22]. Or the LDNS may have to serve its clients over a large geographic area, making it impossible to be close to all the clients. The use of public DNS resolvers such as OpenDNS and Google Public DNS is a good example [23]. A solution to this is to include the client's subnet prefix in the DNS requests [24] so that the authoritative DNS resolver is aware of a portion of the client's IP address, and thus can identify a cache server based on the subnet prefix, increasing the chance that the selected server is close to the client. Obviously, this approach would need substantial changes to DNS and client behaviours.

2. *DNS caching.* During the period of TTL, if there is any update on preferred cache servers, this update cannot be propagated to the LDNS until the TTL expires, leaving clients connecting to sub-optimal servers. This issue can be addressed by setting small TTL values at the cost of sending a large number of DNS queries. Moreover, as shown in [20] the applications such as web browsers may modify and adopt a large TTL value.
3. *Centralized global mapping.* Even with the authoritative DNS resolver aware of client's IP address and having a small TTL value, it would need a great deal of effort to measure the performance of the cache servers and identify the geo-to-IP mappings. This requires global knowledge to analyze user latency data and collect performance and health data of the servers in real-time, creating a scalability challenge. Akamai has been leading the development in this field [25], but this requires a considerable amount of investment in infrastructure and operations [6], making it forbiddingly costly to small or new operators.

As a result, newer CDN operators are increasingly adopting anycast-based CDN architectures.

B. Anycast-Based CDN

The DNS procedure is the same as DNS-based CDNs. The difference is that for an anycast-based CDN, instead of returning a unicast IP address, an anycast IP address is returned by the authoritative DNS server, which is shared by all the cache servers in the CDN. That is, the same IP address is announced from multiple servers/locations. Then the user traffic is routed to the cache server "closest" to the client, based on BGP's notion of best path. This simplicity of anycast mitigates the issues identified above for DNS-based CDNs, making it attractive to new and small CDN operators.

For example, Bing—part of Microsoft Azure—is an anycast-based CDN that consists of FastRoute nodes [4]. Other prominent anycast-based CDNs include CloudFlare [26], Amazon AWS [27] and Google Cloud CDN [28]. The evaluation of Bing CDN demonstrates that it can perform better than DNS-based CDNs, despite its simple design [4]. The work in [29] through passive measurement shows that anycast-based CDNs perform well, and more than 50% of web users access content served by anycast-based CDNs during peak times.

However, the simplicity of anycast comes at a cost—*Loss of control*—how the traffic is routed is entirely at the mercy of BGP routing. Due to this, there are a number of issues.

1. *Cascading failure.* When a cache server is overloaded, the most practical intervention would be to utilize BGP techniques such as AS Path pre-pending or withdrawing routes. This may, however, create cascading failures. An overloaded server withdrawing its route may cause a traffic swarm to a nearby server, which may get overloaded and withdraw its route, and so on.
2. *Unaware of network/server dynamics.* BGP is unaware of network performance or server load, and may direct traffic to a sub-optimal server. In contrast, ISP/CDN operators have the knowledge, but are unable to control where the traffic should be directed, making it impossible to take advantage of ISP/CDN collaboration.
3. *TCP session breaking.* Even if we were able to divert traffic from an overloaded server to an alternative server by manipulating, for example, routing metrics, all the existing TCP sessions would have to be terminated and need to reestablish with the alternative server.
4. *Anycast route flapping.* A route change in the middle of a TCP session can result in a client communicating to an alternative server, causing the ongoing TCP sessions to terminate. This is particularly undesirable for heavy flows such as video streaming applications. The issue of route changes is often used to question the viability of anycast-based CDNs. However, it has been shown the amount of traffic affected by route changes is extremely small [30], [31].
5. *Retrieving from multiple servers.* The nature of anycast makes it impossible to retrieve content over concurrent TCP sessions connected to multiple cache servers (e.g., video streaming over multiple servers). All the sessions have to be connected to the same server.

All these issues are more or less a scalability challenge to the system. Adding more cache servers can mitigate all these problems. However, the intense competition in the content delivery market requires operators to cut costs. Even for hotspots where adding new servers is inevitable, it takes time (weeks or months) to get a new server deployed. Given all these challenges and with the programmability of SDN/NFV and the flexibility of virtualization, we are motivated to design a scalable anycast-based CDN architecture that enables precise and fine-grained control on content delivery.

III. T-SAC Architecture

Anycast-based CDNs generally perform well [4], [5], [29]. In [5], it shows that $\sim$20% of the users are directed to sub-optimal servers. Potentially, we need to redirect some of the 20% users to a more suitable server. However, this does not justify a complicated solution that requires high CapEx/OpEx, a principle followed in our design process.

The deployment of T-SAC does incur the costs of deploying and operating the SDN/NFV infrastructure, that is, the deployment of programmable switches and NFV nodes. Both SDN and NFV were proposed with a focus on, technically building a flexible, programmable and agile network, and financially the reduction in CapEx and OpEx. We do assume that SDN and NFV are coming in one way or another. In a matter of fact,

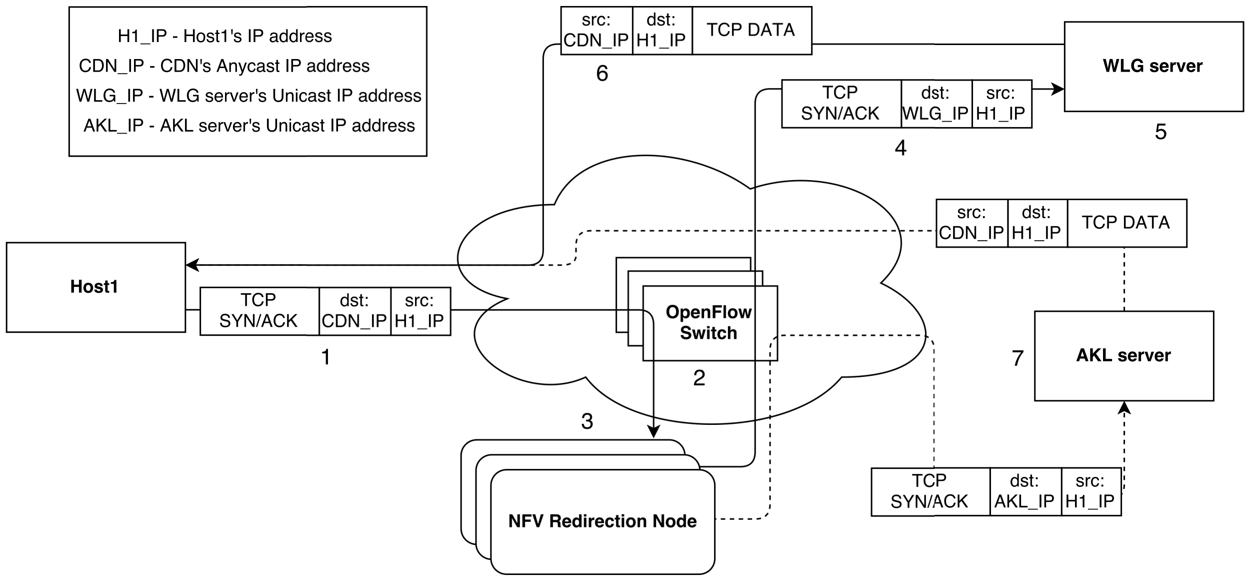


Fig. 1. Architecture overview.

SDN and NFV have been (or being) deployed, for example, Google's B4 [32] and the CORD project [33], which is widely supported by the major ISPs globally. Furthermore, T-SAC does not require a full deployment to function as it can coexist with native anycasting. The SDN switches and NFV nodes can be deployed incrementally, starting from a single switch and NFV node.

To enable precise and fine-grained control and make the system scalable and practical for deployment, there are a number of goals that T-SAC needs to achieve, that is, being able to:

1. *offload a server in a proactive, flexible, dynamic, smooth and non-interruptive manner* – redirect traffic among the CDN servers in a manner that reflects server and network dynamics, while the existing TCP sessions are not affected.
2. *redirect to alternative CDN servers based on ISP/CDN collaboration* – the most suitable servers determined by CDN/ISP operators based on their knowledge of network and server dynamics may be different from the one determined by native anycasting.
3. *provide user transparency* – there is no modification to the user side. The user is unaware of traffic redirection.
4. *minimize the knowledge needed about the server state* – this simplifies the design of the architecture and makes it practical for real-world deployment.
5. *coexist with native anycast and ensure incremental deployment and flexible placement* – since anycast-based CDNs already perform well, a complete overhaul is not justifiable or needed. Being able to coexist with native anycast enables incremental deployment of the system. It is also desirable to allow flexible placement of the system in the network for operational simplicity as well as performance optimization.
6. *mitigate the anycast route flapping problem* – since the architecture makes use of both anycast and unicast addresses of the server, the anycast flapping problem is naturally eliminated for the part of the route using the server's unicast address.

A. System Architecture

Figure 1 overviews the proposed CDN architecture. It gives an example of how anycast CDN traffic is routed. The example system consists of a host making content requests, two anycast CDN servers—one in Auckland (AKL) and one in Wellington (WLG)—and an NFV Redirection Node.

1. Host1 sends a packet destined to the anycast IP address of the CDN (CDN_IP). The packet could arrive at the WLG or the AKL server, since they share the same anycast IP address and are within the same server group. (See Section III-A1 for how servers are grouped.)
2. When the packet enters the ISP network it is passed to an NFV Redirection Node. This is achieved by adding flow rules to the SDN switches in the network that forward the packets to the NFV node. (See Section III-A2 for how the switch and the forwarding rules could interact with NFV nodes. See Section IV for the implementation of the rules.)
3. The NFV Redirection Node accepts the packet, performs DNAT changing the anycast address of the servers to the unicast address of the WLG server (WLG_IP), and then sends it out back into the ISP network. (See Section III-B and III-C for how the server selection decision is made.)
4. The packet originally sent by Host1 now has WLG_IP as its destination address. The packet is then forwarded to the WLG server using normal routing rules.
5. The WLG server processes the packet.
6. In response, the WLG server performs SNAT, using its anycast IP address as the source address (CDN_IP). The WLG server is then able to send its response straight back to Host1, without the involvement of the NFV Redirection Node. The response does not have to travel back through the OpenFlow switch. Host1 will accept this response since it comes from the CDN's anycast address that it sent its

packet to. Host1 has no knowledge that its packet was DNAT'd and sent to the unicast address of a specific server (WLG_IP). (See Section III-C2 for the design choice.)

7. In this instance the AKL server was not communicated with, but it is possible that if the user opened a new TCP connection and the WLG server was overloaded the flow could be redirected to the AKL server. (See Section III-B and III-C for how flows are redirected.)

The above gives a high-level overview of how the system works without going into details. Now, we illustrate how servers are grouped for effective redirection, and the design choice of using the NFV node instead of the SDN switch for redirection and how to manipulate the forwarding rules in the switch to affect the redirection behaviours at the NFV node.

1) Server Group: The servers are grouped so that if traffic needs to be redirected from one server to another in the group, the user would experience little difference in latency or bandwidth capacity. For example, in our implementation we decided to group cache servers in Auckland, Wellington and Christchurch all together. This is because all their locations are relatively close to each other. The difference in latency, when connecting to different servers in the group, would have little impact on the user's QoE.

This is in contrast to FastRoute [4], where to avoid tit-for-tat or ping-pong redirection a user has to be redirected to a server in a different layer, which is significantly further away from the user (Fig. 17), say, from an edge server in Auckland to a regional hub in Singapore, or from a regional hub to a central hub, not the other way around or between peers. For example, in FastRoute it is impossible to redirect from Wellington to Auckland if both are edge servers, let alone the redirection within the same PoP (Point of Presence). Our architecture, however, allows the redirection between potentially any servers.

Note that there can be multiple levels of server groups, using PoP to illustrate, an intra-PoP level where servers are within the same PoP (e.g., servers in Wellington) and an inter-PoP level where servers are located at different PoPs. Inter-PoP level can be further divided into national level (a nation-wide inter-PoP server group) and international level (a global inter-PoP server group). Our architecture has the flexibility to divert traffic within the same level or between levels. For example, T-SAC may first redirect to a server within an intra-PoP server group (e.g., from Server A to Server B in Wellington). If impossible, say, all the server instances in Wellington are overloaded, T-SAC may then redirect within a nation-wide inter-PoP server group (e.g., from Wellington to Auckland) followed by a global inter-PoP server group if needed. This layering approach of server groups in T-SAC not only improves the scalability of the system but also user experience. In comparison, the layering approach in FastRoute [4], although designed for scalability, does not have the level of flexibility that T-SAC has. In practice, the server group could be predefined and/ or discovered with the help of tools such as *iGreedy* [34].

2) NFV Redirection Node: The NFV Redirection Node directs packets to a server's unicast IP address, and then the server replies with its anycast IP address. The redirection process is completely transparent to the user. Note that the redirection functionality could also be implemented between the OpenFlow switch and the SDN controller without the involvement of an NFV node. However, there are a number of reasons to adopt the NFV approach.

1. It keeps the switch (data plane) simple and scalable. Otherwise, the switch has to perform the redirection functionality and maintain a large flow table at the socket level. With the NFV approach, the complexity is moved to the NFV node and the valuable TCAM in the switch is preserved for the packet forwarding process.
2. It gives us the flexibility to implement the network functionality anywhere in the network, even in the switch as a logically independent module if desired (e.g., a software switch). This makes it convenient for operational simplicity and performance optimization.
3. It makes it possible to add new functionality or augment existing functionality without being constrained by OpenFlow. For example, tunneling (instead of DNAT/SNAT) and QoS provisioning could be added. But, they are not supported natively in OpenFlow.

Note that the NFV Redirection Node can be disabled and enabled on the fly. If the redirection is not desirable, the SDN controller may delete the forwarding rule in the switch, which then stops forwarding anycast packets to the NFV node. Any new flows thereafter will be handled by native anycasting. If the redirection becomes desirable, NFV allows us to easily set up an NFV Redirection Node. The SDN controller just needs to insert a forwarding rule in the switch, which then starts forwarding anycast packets to the NFV node.

In terms of the scalability of the NFV node, the operator has the control on what and how much traffic is handled by a particular NFV node by manipulating the forwarding rules in the switch. No coordination or communication is needed between the NFV nodes. For example, the operator could mandate through the forwarding rules that only the traffic to CDN A or only the traffic from Subnet A is to be handled by the NFV node. Other traffic will then be handled by native anycast or other NFV nodes. This means that any scalability problem regarding the NFV node is controllable. If an NFV node fails, only the involved TCP sessions get interrupted. However, these sessions can restart with the native anycast, or a new forwarding rule could divert the traffic to an alternative NFV node. To avoid conflicting rules (e.g., one rule says "Forward to Node A" while another says "Forward to Node B"), intent framework and verification techniques such as SNAP [35] and VeriFlow [36] could be used. Also note that the data packets from the server are routed directly to the user without having to go through the NFV node. Only the TCP SYN and ACK packets need to go through the NFV node. So, for a particular flow the NFV node only handles a small amount of traffic.

B. NFV Redirection Architecture

The NFV Redirection Node (Fig. 2) is a core part of T-SAC, and can be easily spun up and placed strategically in the network due to the nature of NFV [37]. If a server is overloaded, the NFV node is responsible for maintaining

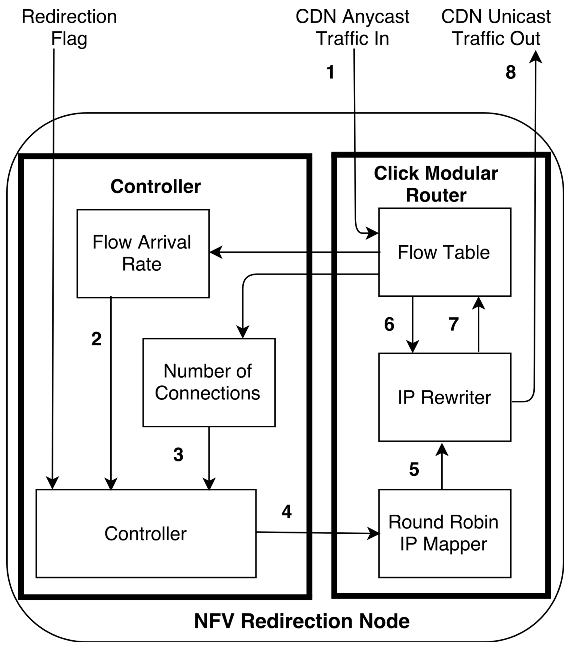


Fig. 2. NFV redirection architecture.

existing flows with the current server and redirecting new flows to a suitable alternative server. The NFV node consists of two main components, the Click Modular Router [14] and the Controller. The Click router follows the rules that determine when and where new flows should be redirected. These rules are created and updated by the Controller via an API provided by Click (Fig. 2 step 4).

1) Controller: The Controller is the brain of the NFV Redirection Node. It makes the decisions regarding when and where traffic should be redirected. It also decides how much of a server's traffic should be redirected. In the current implementation we use the number of connections that a server has as the metric to measure the server load. This is useful to emulate possible TCP SYN flood DDoS attacks. In practice, other metrics could also be used, e.g., CPU, memory or network bandwidth usage depending on the type of traffic hitting the server. The Controller is fed with the information about the flow arrival rate of the server and the number of flows that the server has, which is retrieved from the Flow Table. Such information could also be retrieved from the server. Before this information is passed to the Controller, it travels through two modules: Arrival Rate and Number of Connections, which transform the Flow Table information into meaningful metrics that the Controller can use (Fig. 2 steps 2 and 3). In addition, the Controller is fed with no-redirect flags, which indicate which servers are not accepting redirected traffic. The number of flows that a server is handling may change rapidly over time. To smooth out the spikes and thus avoid overreacting, we used Exponentially Weighted Moving Average (EWMA) [38] to calculate the number of flows. We set the EWMA weight to 90% to make it reasonably responsive to traffic changes.

Our design leverages ISP/CDN collaboration [20], [39]–[42], in particular, the informed user-server assignment incorporated in NetPaaS [40]. The Controller may interact with such platforms and utilize transit routes [43] to identify the most suitable cache server.

2) Redirection Algorithm: When the Controller decides that traffic should be redirected away from a server it creates a Redirection Ratio for that server. This redirection ratio is the ratio between the traffic to be accepted and the traffic to be redirected. So in a 2:8 ratio, 8 in every 10 new flows will be redirected. This ratio is passed to the Round Robin IPMapper (Fig. 2 step 4) in the form of a string. A 2:8 ratio string may look like the one in Fig. 3.

This string shows 2 connections that stay with the server 10.0.1.1, and the other 8 connections being redirected to 10.0.1.2 or 10.0.1.3. For the string to be created neither the server 10.0.1.2 nor 10.0.1.3 had their no-redirect flag set, meaning they are willing to accept redirected traffic. The Controller takes the no-redirect flag, flow arrival rate and the current number of connections that a server has as its inputs. The Controller then uses these inputs as its variables in the following equation to create the Redirection Ratio:

$$Redirection\ Ratio = (\lambda \times \tau - \epsilon) : (N_C - N_T + \epsilon) \quad (1)$$

where λ is the flow arrival rate, τ is the sampling interval, ϵ is the change in the number of flows that the server is handling during τ, N_C is the number of flows the server currently has and used to derive ϵ, and N_T is the threshold that new flows should be redirected at.

The left side of the ratio defines the portion of the new flows that should stay with the BGP determined server, while the right side of the ratio determines the portion that should be redirected. The left side is determined using the number of new flows arrived during the sampling interval ($\lambda \times \tau$) minus the change in the number of flows that the server is handling observed during the sampling interval (ϵ). This indicates that if the server has just seen a decrease in the number of flows that it is handling, it is in a better state to accept more traffic. Whereas if it is an increase, the server may not want to keep too much of the new traffic. The right side of the ratio uses the number of flows that the server currently has (N_C) minus the threshold that traffic should be redirected at (N_T), plus the change seen in the number of flows during the sampling interval (ϵ). When this side of the ratio is greater than 0, traffic redirection is activated. That is, we expect the server load during the current sampling period to be greater than the max number of flows that the server wants to handle (the threshold, N_T). We redirect this portion of new flows so that the server load does not exceed the threshold (N_T). We use the change in the number of flows observed during the previous sampling period (ϵ) to try to pre-empt the threshold being broken. The redirection ratio is adjusted on the fly according to server dynamics, λ and N_C.

3) Click Modular Router: The Click Modular Router is a powerful software router that can be easily configured and modified to perform many different tasks. It is a good example of what the idea of NFV is capable of. While this is a piece of software, it is capable of performing the same task as a full network router. But where a physical router must be fixed in one place, a Click router can be spun up on almost any Linux box or hardware in the network. In the Click Modular Router,

```
--10.0.1.1-10, --10.0.1.1-10, --10.0.1.2-10, --10.0.1.2-10, --10.0.1.2-10, --10.0.1.2-10, --10.0.1.3-10, --10.0.1.3-10, --10.0.1.3-10, --10.0.1.3-10
```

Fig. 3. A 2:8 redirection ratio string.

<Source IP>	<Source Port>	<Dest. IP>	<Dest. Port>	=>	<Source IP>	<Source Port>	<new_Dest. IP>	<Dest. Port>
10.0.0.1	20125	10.10.10.10	8080	=>	10.0.0.1	20125	10.0.10.1	8080

Fig. 4. Click flow mapping.

the Round Robin IPMapper accepts the ratio string from the Controller (Fig. 2 step 4). It then, when the IPRewriter asks for an IP address for redirection, returns in round-robin order the next IP address in the string (Fig. 2 step 5).

The Flow Table is another important part of the module. It stores a flow mapping for each TCP flow destined for a CDN server that enters the NFV Redirection Node. A flow mapping consists of two TCP 4-tuples, one before and one after the redirection to be used by the flow, as seen in Fig. 4. Only the Destination IP needs to be changed in the mapping from the anycast address to the unicast address of the server. The Source IP and Port will stay the same so that the host can receive the server's response. The destination port remains the same as we are still retrieving the same content from the same application.

4) No-Redirect Flag: One of our goals is to direct a user to a server that has the capacity and willing to handle additional traffic. The no-redirect flag helps us achieve this. To identify such servers, some state information about the servers is needed. However, passing state information around complicates system design and creates a lot of communication overhead. Meanwhile, the benefits may not be achieved if the state information is not handled in a timely and reliable manner. Moreover, the Controller will need to be updated frequently. This is not a scalable approach. To minimize the amount of state information needed for the Controller, we introduce a single-bit no-redirect flag. The CDN servers use this single-bit no-redirect flag to indicate whether it is willing to accept redirected traffic. This single bit could be added to control messages that are passed around the CDN network, e.g., the messages indicating the health status of the servers commonly implemented in CDNs. Or, a dedicated interrupt message could be implemented. With the no-redirect flag, the state information needed for the controller is reduced to a single bit. The server may use any metrics to set the no-redirect flag, e.g., the number of flows that the server is currently handling, CPU utilization, network bandwidth, or maintenance status.

C. Redirection in Operation

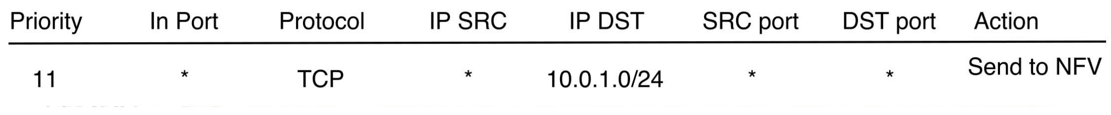
Having introduced the elements of the architecture, we now focus on how redirection works in operation.

1) Redirection Procedure: The CDN anycast traffic is forwarded by the OpenFlow switch to the NFV Redirection Node. Each incoming packet has its 4-tuple checked against the existing flow mappings (Fig. 2 step 1). If there is a match, the packet belongs to an existing flow. The packet is then handled by the IPRewriter, and gets its anycast IP address changed to the unicast address of the server according to its flow mapping (Fig. 2 step 6). After that, the packet is sent out back to the ISP network (Fig. 2 step 8). If no match, it is the first packet of a new flow. A new flow mapping needs to be created. The packet is then handled by the IPRewriter while awaiting for the new flow mapping to be created (Fig. 2 step 6). The IPRewriter gets the unicast IP address of the selected server from the IPMapper (Fig. 2 step 5) and creates the new mapping. The packet gets its anycast IP address changed to the unicast address of the server according to the new mapping. The packet is then sent out back to the ISP network (Fig. 2 step 8). Meanwhile, the IPRewriter inserts the new flow mapping into the Flow Table (Fig. 2 step 7).

The IP address of the selected server provided by the IPMapper (Fig. 2 step 5) is determined by the Redirection Ratio string, which is obtained from the Controller (Fig. 2 step 4). The Redirection Ratio string is created by the Controller based on the redirection algorithm, no-redirect flag and server load and dynamics, which is provided by the Arrival Rate and Number of Connections modules (Fig. 2 steps 2 and 3).

2) Redirection Method: To make the redirection process transparent to users, tunneling or IP Rewriting may be used. Tunneling has some performance advantages over IP Rewriting as TCP checksum has to be recalculated after an IP address is rewritten. However, tunneling is not supported natively in OpenFlow. A workaround has to be carried out in the switch. Therefore in the current implementation, we chose IP Rewriting for redirection. Then, the issue is that in its response, the server will need to have its source address changed from its unicast address to its anycast address, as the user expects a packet with the server's anycast address. Moreover, due to this process TCP checksum has to be performed over every single data packet, adding a performance penalty.

To address these issues of IP Rewriting, DNAT is performed on the SYN and ACK packets—rewriting from the server's unicast address to its anycast address—just before they are passed to the server application. As a result, the data packets from the server are routed directly to the user without having to go through the NFV node. Only the TCP SYN and ACK packets need to go through the NFV node and undergo IP Rewriting. The performance penalty of TCP checksum on these small packets can be neglected.



Priority	In Port	Protocol	IP SRC	IP DST	SRC port	DST port	Action
11	*	TCP	*	10.0.1.0/24	*	*	Send to NFV

Fig. 5. OpenFlow rule.

TABLE I
BBB RESOLUTION AND BITRATES (Kbps)

Resolution	320×180	480×270	640×360	768×432	1024×576	1280×720	1920×1080	3840×2160
Bitrates	200, 400	600	800, 1000	1500	2500	4000	8000	12000

IV. SYSTEM IMPLEMENTATION

Mininet [15] was used as the emulation environment to implement T-SAC. OpenFlow [11] and Ryu [13] controller were used as the implementation of SDN. We then only need to add a rule to the OpenFlow switch so that CDN anycast traffic is forwarded to an NFV node. An example of a rule used in the implementation can be seen in Fig. 5, showing eight of the main parameters that an OpenFlow forwarding rule can have. This rule says that any TCP packet destined for the CDN using its anycast IP address should be forwarded to the NFV node. Since the server responds with its anycast IP address the reply packet from the server does not have to be forwarded to the NFV node. As a result, only one user-defined OpenFlow forwarding rule needs to be added to the switch in this case. In practice, multiple rules could be added on the fly to specify what traffic is to be handled by which NFV node.

We used Click [14] to implement the NFV Redirection Node. In practice, ClickOS [44] and In-Net [45] could be used to speed up Click and address security issues. The capacity of the NFV node is essentially limited by the underlying NFV platform. It shows that ClickOS [44] can deliver production-level performance for carrier-grade NAT operations, a functionality used by the NFV nodes. Note that the NFV node only needs to handle TCP SYN and ACK packets. The data packets are routed directly from the server to the user. Furthermore, the load of each NFV node can be managed by manipulating the forwarding rules in the OpenFlow switches. In other words, the NFV nodes only need to handle the load that they can handle, and leave the rest to native anycasting. There are also some other NFV platforms being proposed such as Eden [46], which allows the efficient support of network functions at network's edge. Eden is an alternative to realise the NFV nodes. It is worth noting that the scalability of T-SAC is achieved due to a number of factors, namely, SDN and NFV scalability, incremental deployment and flexible placement, and the layering approach of server groups. Please see the discussions in Section VI for details.

The CDN servers are equipped with HTTP web servers implemented over node.js [17]. To emulate the performance of T-SAC, these servers all hold the same content, ranging from some random images to DASH videos. One DASH video was Big Buck Bunny (BBB) [47] encoded to 10 different bit rates. This allows us to show the improvement using our architecture on the retrieved bitrate encoding. The standard dash.js library [48] running in a chromium browser [49] was used to retrieve the DASH video. Table I shows the bitrate levels for different resolutions as defined by DASH specifications [19].

When making requests for the images hosted on the CDN server, we used Wget [18]. Wget makes HTTP requests just like a browser does. The key exception is that when retrieving a web page that has multiple pieces of content, e.g., embedded images. A web browser such as Chrome or Firefox may create up to six TCP connections and retrieves each image in parallel, whereas Wget retrieves all the images using a single TCP connection. For the purpose of testing, there is little difference in using a browser or Wget. However, Wget is a non-interactive command line tool, so it can be easily called from our scripts. Traffic was generated based on Poisson distribution. We set the mean generation time to an interval that would give us the desired number of flows. For example, using (2), if we know the average time to retrieve the content from the server is 10 seconds (t), and we want to have an average of 500 open TCP sessions (N_C) at a time, we would use a mean interval of 0.02s ($1/\lambda$, λ being the flow arrival rate) to generate flows.

$$1/\lambda = N_C/t \tag{2}$$

V. EVALUATION

Through evaluation, we would like to answer a few questions, that is, can T-SAC:

- redirect traffic with the popular web and streaming applications?
- redirect the right amount of traffic, handling well in the presence of slow movement of traffic as well as sudden bursts of traffic?
- redirect at the right time so that the server load does not overshoot the limit?
- prevent unnecessary redirection and ensure the preventing mechanism behaves properly?

To this end, two overloading methods are used in the evaluation. The first one is to generate a large amount of traffic using Wget to retrieve content from web servers. This amount of traffic is greater than what a single server would be able to handle. The second one retrieves a dash.js video from a server with limited bandwidth. We would like to see the performance of T-SAC on the retrieved bitrate for each chunk of the video.

A. Network Setup

All the experiments were carried out in a Mininet emulation environment running on an AWS EC2 instance [16], using

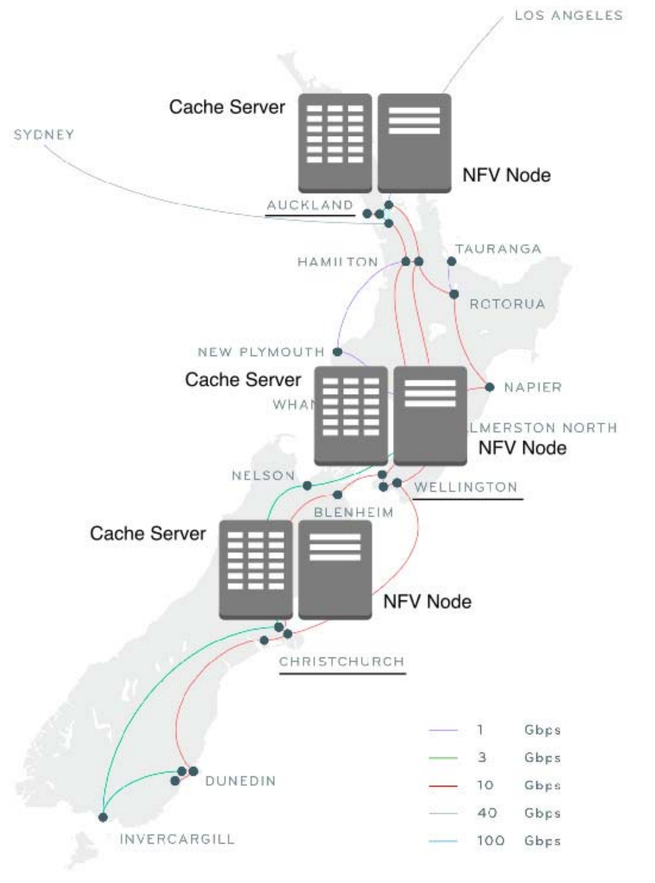


Fig. 6. Evaluation topology (courtesy: REANNZ ISP network [50]).

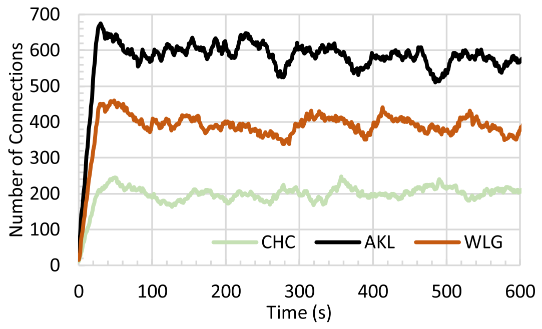


Fig. 7. No redirection.

the topology of REANNZ ISP network shown in Fig. 6. The clients, servers and NFV Redirection Node all ran on their own Mininet host. Nowadays, the CDN servers have multiple terabits of capacity, able to handle millions of flows. As our implementation is emulation based, it is not suitable for stress testing or handling millions of flows. The purpose of the evaluation is to demonstrate T-SAC's capability of diverting traffic in a proactive, dynamic, smooth, non-interruptive and flexible manner, echoing the evaluation questions raised at the beginning of the section. For this purpose, we only need to create a server load that can clearly demonstrate the redirection capability of our architecture, instead of a load for stress testing.

It is also worth noting that the evaluation would be much more interesting and realistic if we had an actual deployment of T-SAC or were able to obtain traffic traces from ISP/CDN operators. We would then be able to analyze the impact of server load, server geographic location and network dynamics in meaningful real-world scenarios, or investigate the impact of PoP-to-PoP communications via transit routes vs. a backbone network. These studies would help ISP/CDN operators determine the grouping of servers (see Section III-A1 for details) or the impact of redirection in practice. We will leave these for future studies.

For the evaluation using Wget, the redirection threshold is set to 500 flows. This parameter defines at what point the traffic should be redirected away from a server. We chose 500 as the threshold for the purpose of evaluation, demonstrating the features of our architecture. In practice, each server may run on different hardware with varying bandwidth connections to the wider ISP network. The threshold should be left up to the choice of the CDN operators. They are in the best position to determine at what point traffic should be redirected. For the evaluation using dash.js video streaming, which may require a substantial amount of bandwidth to function, bandwidth was the chosen metric to determine when traffic should be redirected.

B. No Redirection

In the first experiment, we would like to show that T-SAC is able to successfully route traffic to servers without redirection. To show this, we set up three CDN servers, one in Wellington (WLG), one in Christchurch (CHC) and one in Auckland (AKL), which are expected to handle around 200, 400 and 600 open connections, respectively, at a time throughout the experiments. Redirection is disabled. The results are shown in Fig. 7. As expected, three servers in WLG, CHC and AKL are processing an average of 200, 400 and 600 connections throughout the experiments. In this set of experiments, the OpenFlow switch has a forwarding rule which forwards anycast traffic to the NFV Redirection Node which then directs the packets to individual CDN servers. In another set of experiments (not shown in this paper), the forwarding rule is disabled – the anycast traffic is forwarded by the OpenFlow switch directly to the individual servers. Similarly, the three servers are handling an average of 200, 400 and 600 connections, respectively. This indicates that the architecture is able to direct traffic through the NFV node or relying on the native anycasting.

C. Baseline Redirection

In this scenario, we show an example of baseline traffic redirection. The redirection threshold is set to 500 connections. The AKL server is expected to experience a high traffic load with 600 connections being destined for it, while the WLG and CHC servers are expecting 200 connections each. This means that some of the 600 connections need to be redirected away so that the load handled by the AKL server does not significantly go beyond 500 flows. Fig. 8 shows that around 500 flows are actually being processed by the AKL server throughout the experiment. This is because the NFV node redirects traffic when the number of flows to the AKL server exceeds or is

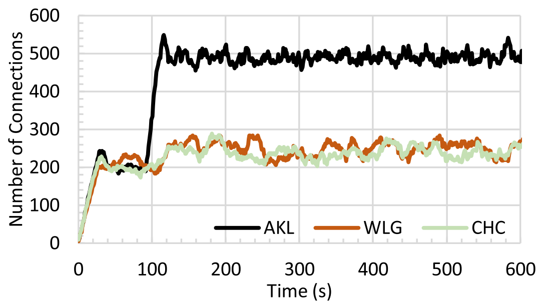


Fig. 8. Baseline redirection.

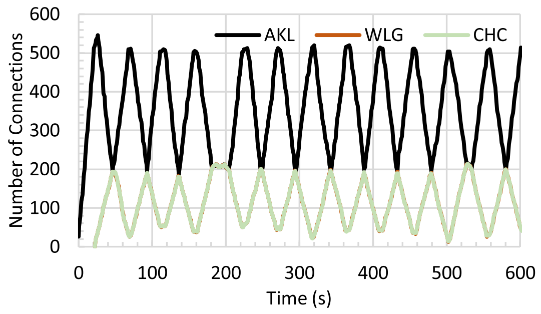


Fig. 9. Simple bound redirection.

about to exceed 500 flows. When this happens the NFV node sets a ratio regarding how much traffic to be accepted and redirected. This ratio is re-calculated every second so that the node is able to react to any sudden changes to the arrival rate and/ or departure rate of new and existing flows. The new flows to be redirected are sent to the other servers in the server group, which in this example are the WLG and CHC servers. Each of these two servers receives about half of the redirected flows in an attempt to balance the load among the servers. This results in the two servers receiving an average of 50 redirected flows each or in total 250 flows each.

Figure 9 shows the redirection behaviours resulted from MIMA proposed in [9] which employs hysteresis-based redirection control. In the figure, the upper bound of the hysteresis is set to 500, the same as the threshold in Fig. 8. The lower bound is set to 200. But the architecture is unable to redirect traffic in a dynamic way. In the figure, the traffic is redirected at a fixed ratio of 1:2 until the number of flows the AKL server is processing drops below the lower bound of 200, at which point the AKL server stops redirecting new flows. It then starts to redirect traffic again after it breaches the upper bound of 500 flows. In this example, the architecture redirects more traffic away than necessary, and thus is not able to achieve an equilibrium load level. This may result in two potential issues. The first one is that the WLG and CHC server may be put at the risk of being overloaded. The second is that some of the redirected users could have received decent performance from the AKL server, but now have to be redirected to either WLG or CHC and may suffer performance penalty. In contrast, our architecture only redirects the right amount of traffic to alternative servers.

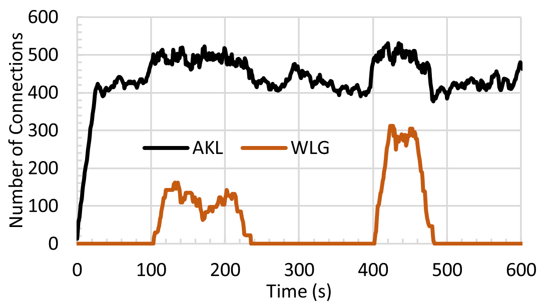


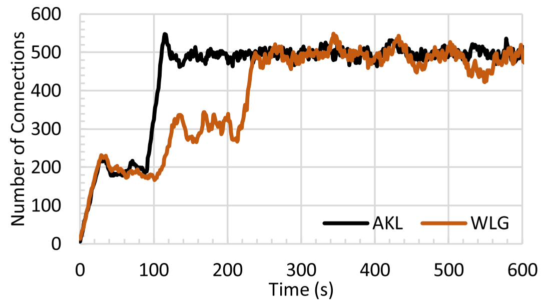
Fig. 10. Redirection with sudden bursts.

D. Redirection With Sudden Bursts of Traffic

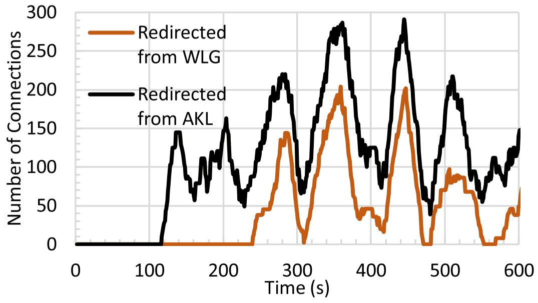
Figure 10 shows how T-SAC reacts to large sudden changes in flow arrival rate that a server may experience. In the figure, the AKL server is initially receiving appropriately 400 TCP flows, while the WLG server has no traffic generated for it. In the time periods 90-240s and 400-480s the AKL server sees large spikes in flow arrival rate, with 600 and 800 flows, respectively, destined for the AKL server. Even with these large spikes in traffic we do not see the AKL server handling a load significantly over its threshold of 500 flows. The additional traffic is quickly and smoothly offloaded to the WLG server. This is largely due to the redirection procedure being proactive, dynamic and smooth. When the load is approaching to the threshold, the redirection is started proactively (proactive). As the flow arrival rate starts increasing rapidly, the redirection ratio is adjusted accordingly to quickly divert traffic to the alternative servers (dynamic). However, T-SAC does not over redirect. It only redirects the right amount of traffic, trying to keep the load of the current server below the threshold (smooth). This ability of responsiveness is an important feature of our architecture when we consider the impact of social media events such as flash crowds, where a huge number of people request the same content at almost the same time [51].

E. Preventing Tit-for-Tat Redirection

In this experiment, we look at the scenario where all the CDN servers in the group are highly loaded and are not in the position to accept redirected traffic. Fig. 11 shows two servers AKL and WLG with each server reaching the redirection threshold of 500 flows. In the figure, both servers start off handling around 200 flows. After 90 seconds, the AKL server starts to see an increase in traffic, with 600 connections destined for it. When the AKL server reaches the threshold of 500, the additional flows are redirected to the WLG server. This can be seen in the figure between 120 and 200 seconds where the WLG server is receiving around 100 flows from the AKL server (Fig. 11b). Later at around 220 seconds the WLG server sees an increase in traffic. This increases the amount of traffic originally destined for the WLG server from 200 to 400 flows. Plus the 100 flows redirected from the AKL server, the load is now at the threshold of 500 flows. Traffic starts to be redirected from the WLG server to the AKL server. This can be seen at time 240 in Fig. 11b. At the same time more traffic is redirected from the AKL server to the WLG server, as more

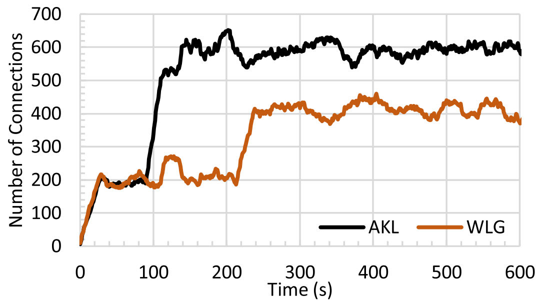


(a) Number of open TCP connections

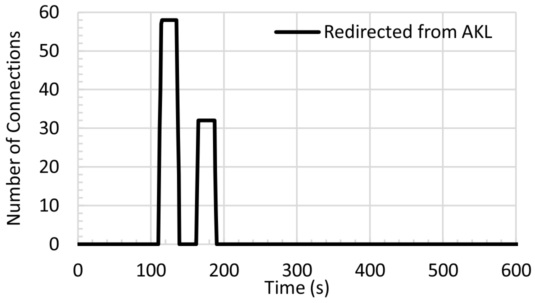


(b) Number of redirected connections

Fig. 11. Tit-for-Tat redirection.



(a) Number of open TCP connections



(b) Number of redirected connections

Fig. 12. Redirection with no-redirect flag.

traffic is redirected from the WLG server to the AKL server. This causes a tit-for-tat redirection between the AKL and WLG servers. While in this example the traffic is balanced between the two servers, it is inefficient and unnecessary to redirect traffic in this scenario. It ends up putting both servers at their threshold and ultimately providing users with no real benefit.

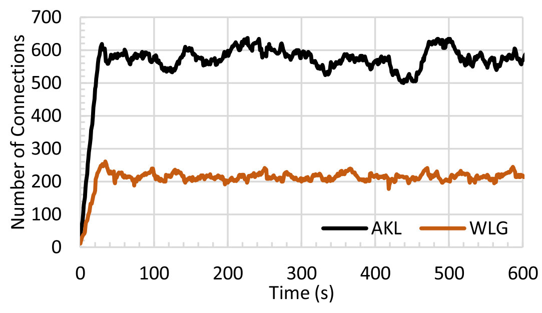
In order to avoid this tit-for-tat redirection, the no-redirect flag is used. The servers are now able to set their no-redirect flag. In this example a server sets its no-redirect flag whenever the number of flows it is handling is greater than 200 – at this point the server will not accept any redirected new flows. Fig. 12 shows the results of using the no-redirect flag.

Both servers start off with around 200 flows. After 90 seconds, the AKL server starts to see an increase in traffic with 600 flows destined for it. Since the WLG server is handling more than 200 flows at the time and has its no-redirect flag set, the AKL server cannot offload any of its new flows. At time 110 the flag is unset and the AKL server starts to redirect some traffic to the WLG server. Since this pushes the number of flows to the WLG server well above 200 flows, the no-redirect flag is set. This then happens again 60 seconds later. In the figure, the flat caps in the number of redirected connections indicate that during this period of time no redirected flows arrive or leave, so the number of redirected flows stays the same. At around time 220 the WLG server sees an increase in non-redirected traffic. This increases the traffic originally destined for the WLG server from 200 to 400 flows. So now both servers have their no-redirect flag set – no traffic for the AKL server can be redirected to the WLG server. This leaves each server to process its own traffic, that is, 600 flows for the AKL server and 400 flows for the WLG server. While this does not balance the traffic between the servers like in Fig. 11b, it does stop traffic being redirected to a server that cannot or does not want to handle it. This is important, because we do not want a server at its limit handling redirected traffic, while at any moment a large amount of traffic of its own may arrive. Note that to avoid oscillating redirection FastRoute [4] only allows one-way redirection from an outer-layer server to an inner-layer server (Fig. 17). In contrast, with the no-redirect flag our architecture allows redirection potentially between any servers.

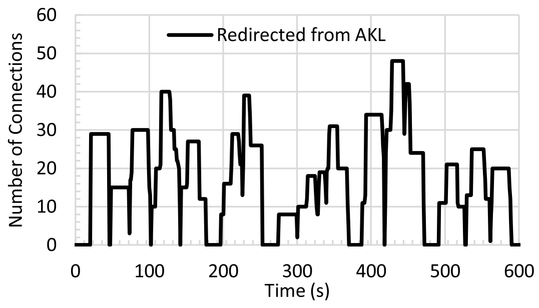
F. Handling No-Redirect Flag

Deciding when the no-redirect flag should be set and unset turned out to be not as simple as depicted in the previous scenario. We initially implemented the no-redirect flag to be set on whenever the number of flows was above a given threshold. A problem arises when the threshold is close to the number of flows originally destined for the server. For instance, the WLG server has a threshold at 200 for its no-redirect flag, while it is receiving an average number of 200 flows without the redirected traffic from AKL. This results in the no-redirect flag of the WLG server flapping on and off, as seen in Fig. 13. It is desirable to avoid this flapping problem, as it provides no benefits and adds burdens to the NFV node and the server.

To mitigate the flapping problem, we tried to smooth out the number of flows that the server is handling by EWMA to determine when to set the no-redirect flag. This did reduce the number of times that the no-redirect flag turned on and off. But, there are still a significant number of flaps. To further address the issue, a hysteresis-based algorithm was developed. The algorithm sets a lower and an upper bound. At the lower bound, the no-redirect flag is turned off when the number of flows drops below the threshold. At the upper bound, the flag is turned on when it breaks above the threshold. The results are shown in Fig. 14. In this figure, the no-redirect flag is

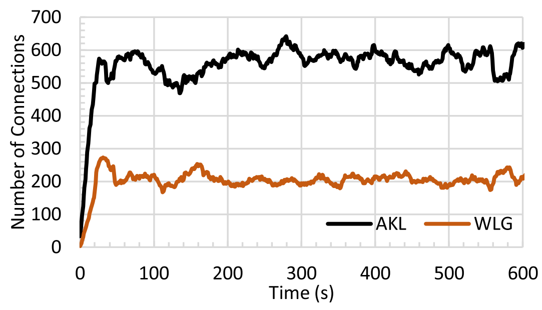


(a) Number of open TCP connections

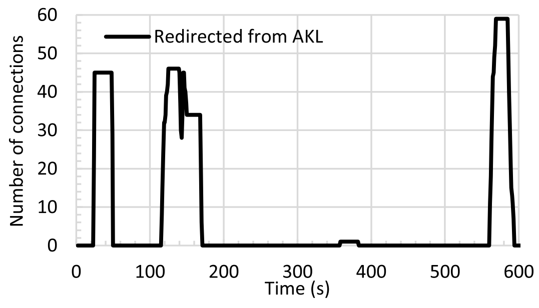


(b) Number of redirected connections

Fig. 13. Instantaneous no-redirect flag.



(a) Number of open TCP connections



(b) Number of redirected connections

Fig. 14. Hysteresis no-redirect flag.

set when the number of flows is above 225, and is only unset once the number of flows drops below 175. Using this method the number of flaps is dramatically reduced. In practice, CDN operators may set appropriate upper/lower bounds based on their knowledge of network and traffic dynamics.

G. Video Streaming

This experiment shows how T-SAC can improve the performance of video streaming applications. We used DASH (Dynamic Adaptive Streaming over HTTP) [19] as our video streaming client. DASH allows the bitrate of the video to be automatically adjusted depending on what bandwidth is available to the client. The bitrate increases or decreases with the bandwidth. DASH breaks up the video file into many small chunks, with each chunk being encoded at several different bitrates. In this evaluation, we set up a set of DASH clients and two CDN servers with the same video encoded at multiple bitrates. We then limited the bandwidth that was provided to each of the CDN servers. Fig. 15 shows the performance of our architecture with and without traffic redirection. We can see that with redirection the average bitrate that the clients received was significantly improved. This is because whenever the number of connections to a server went past the redirection threshold, some of the new flows were redirected to another server. As a result, the existing flows are not significantly affected by the new flows – the current server only accepts the amount of traffic that does not affect the user experience of the existing flows. Meanwhile the new flows redirected away from the current server are connected to an alternative server that has the bandwidth to accommodate these new flows. In this example, we only had two CDN servers. If we increased the number of servers that the new flows could be redirected to, there would be much more space to improve bitrate performance.

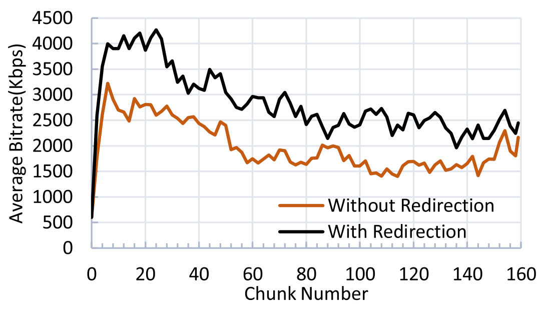


Fig. 15. Average chunk bitrate.

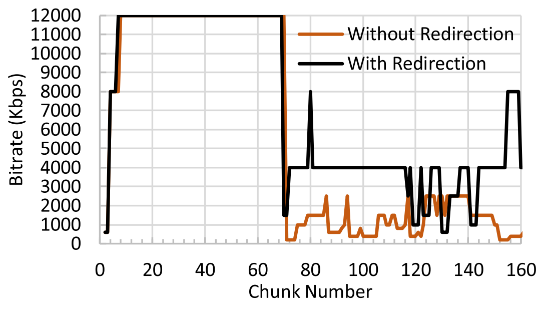


Fig. 16. Chunk bitrate.

Figure 16 shows the bitrate retrieved for each chunk over a particular flow. For the first half of the streaming the server has enough bandwidth for the dash client to retrieve the chunks with the highest quality. After that the server and the network suddenly get loaded, causing the DASH client to retrieve with a much lower bitrate. In response to this, our architecture rapidly redirects some of the new flows to an alternative server. As a result, the bitrate for the DASH client is quickly recovered to the best level that the current server can offer. Without redirection the server is forced to accommodate all the streaming requests. The DASH client has to maintain a low bitrate for the chunks. With the proposed architecture the DASH client can maintain a high bitrate for most of the time. There are some fluctuations in the chunks around 130. This is due to the current implementation of DASH. If QDASH [52] was used, which allows a gradual change in bitrate levels, the fluctuations would be mitigated. SDNDASH [53], an architecture that uses SDN for resource allocation and management for DASH, could be easily built on top of our architecture for fine-grained QoS provisioning.

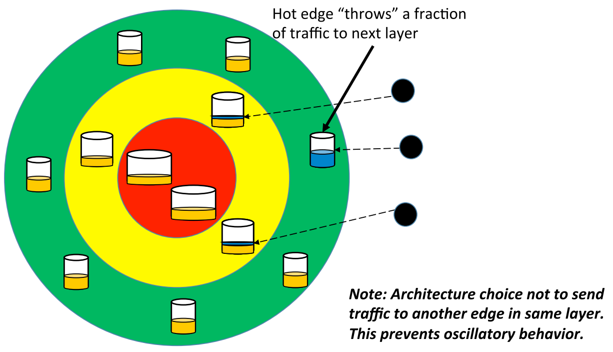

Hot edge "throws" a fraction of traffic to next layer
Note: Architecture choice not to send traffic to another edge in same layer. This prevents oscillatory behavior.

Fig. 17. FastRoute redirection (courtesy: FastRoute NSDI slides [54]).

VI. Discussions

How Are the Design Goals Met?

Goal 1. The redirection being proactive, flexible, dynamic, smooth and non-interruptive will be discussed separately.

Goal 2. Suitable servers are selected based on ISP/CDN collaboration [40], the use of server group [34] and no-redirect flag, and transit routes [43]. While ISP/CDN collaboration is highly recommended, it is not required to make T-SAC work. Without the collaboration from ISP, we would still have the 1-bit flag from the alternative servers and the basic load information from the current server. But, we could only divert traffic at CDN edge, which might not be efficient.

Goal 3. The redirection is done via IP rewriting with no modifications to the client.

Goal 4. The system only needs to know a one-bit no-redirect flag of the servers and the basic load information of the current server. However, if more bits were used to indicate the state of the alternative servers, T-SAC would be able to divert traffic more intelligently, for example, reducing the possibility of overloading or unbalancing the alternative servers.

Goal 5. If not needed, the redirection can be disabled by removing the forwarding rule in the switch. Any new flows thereafter will be handled by the native anycast. Deployment and placement issues will be discussed separately.

Goal 6. Because of the use of the unicast address of the CDN servers after the redirection procedure, the path from the NFV Redirection Node to the server is not affected by the anycast route flapping problem.

Proactive, Dynamic, Smooth, Non-Interruptive and Flexible Redirection: By measuring server dynamics (load and its trend), a redirection ratio is figured out based on the algorithm in (1).

Proactive: If the predicted load is going to break the threshold (server at the risk of being overloaded), redirection is activated and traffic is proactively redirected away.

Dynamic: The redirection ratio is adjusted on the fly according to server dynamics. Rapid increase in traffic may result in a large portion of (or all) the traffic being redirected away. Slow increase may result in a small portion being redirected.

Smooth: Based on the redirection ratio, only the right amount of traffic is redirected away. The goal is to keep the load of the current server just around the threshold, which is considered the limit to provide satisfactory performance.

Non-interruptive: The system only redirects new flows and thus existing flows stay with the current server without being interrupted during the redirection.

Flexible: The use of no-redirect flag allows T-SAC to redirect to any servers without having the problem of servers redirecting to each other. This is in contrast to FastRoute [4], which to address the problem only allows redirection from an outer layer to an inner layer, that is, from an edge server to a regional hub or from a regional hub to the central hub, not the other way around or between peers (Fig. 17). T-SAC may first redirect to a server within the same server group, e.g., from Wellington to Auckland, and then if impossible, redirect to the regional or the central hub, e.g., from Auckland to Sydney or Los Angeles (Fig. 6).

Why Not to Redirect Existing Flows? Firstly, it is unlikely that T-SAC would need to redirect existing flows. Our redirection is proactive, pre-empting the load threshold being broken. Before the server gets overloaded, the redirection of the new flows would already start if the server was at the risk of being overloaded. It shows in Fig. 10 that even with sudden bursts of traffic, the load of the current server stays around the threshold. Secondly, it is not trivial to redirect existing flows without breaking them [42]. It involves the transfer of TCP state information, which is a complex process, making the system unscalable. Research shows that only a small fraction of users need to be redirected [5]. The cost of redirecting existing flows cannot be easily justified. T-SAC only needs to redirect new flows to keep the load of the server under control. This is much more practical than redirect existing flows.

Retrieving Content From Multiple Servers: For applications that support the use of concurrent TCP sessions, T-SAC allows to fetch content from multiple servers simultaneously – concurrent TCP sessions can be established with different cache servers. This may potentially improve the performance of web browsers and streaming applications such as DASH.

SDN and NFV Scalability: The traffic redirection could have been done by the switch. However, this could potentially add a significant amount of load to the data plane. The use of the NFV node essentially offloads the redirection functionality as well as potentially a large number of flow entries from the switch. This results in a simple and scalable data plane.

It is the ISP/CDN operators' decision in regard to what and how much traffic to be handled by an NFV node for the redirection process. This can be easily done by manipulating the forwarding rules in the OpenFlow switches. Moreover, for a particular flow the NFV node only handles the TCP SYN and ACK packets. The data packets go straight from the server to the user. In addition, the NFV nodes do not need to communicate with each other. This means that the scalability issues of the NFV node are controllable, and even with a single NFV node there are no issues such as single point of failure (the node would only handle a fraction of traffic; a failure would only affect this fraction of traffic which could then be reestablished through native anycasting).

Incremental Deployment and Flexible Placement: The nature of NFV means that the NFV Redirection Nodes can be easily spun up or shut down and placed strategically in the network. By manipulating the forwarding rules in the OpenFlow switches, we can manage the load of the NFV nodes. For instance, assuming that there was only a single NFV node deployed, we would only forward the amount of traffic that the single node could handle. The flows forwarded to the NFV node might be redirected to potentially a more suitable server and thus receive better QoS. The flows not forwarded to the NFV node would be handled by native anycasting and directed to the "closest" server. This means that T-SAC supports incremental deployment. This is in contrast to PIAS [7] or FastRoute [4], which needs full deployment (a sufficient number of nodes) before their system can function.

In theory, the NFV nodes can be placed anywhere in the network between the switch and the server. In two extreme cases, the NFV nodes may be collocated with every switch or server. Collocating with the switch makes it harder to get server information in a cost-effective and timely manner, but it is more efficient in terms of redirecting flows among the servers. Collocating with the server has the advantage of easy access to server information but is less efficient when redirecting to another server is needed. In reality, it is a trade-off between these two extreme cases. CDN or ISP edge is the natural choice of placement. In our implementation, we chose CDN edge for the convenience of access to server information.

Why Is T-SAC Scalable? Based on the discussion on SDN and NFV scalability, it is unlikely that the data plane of the SDN switches or the NFV redirection nodes may get stressed. The redirection functionality is moved from the data plane to the NFV node. The load of the NFV node can be managed / controlled by manipulating the forwarding rules in the OpenFlow switch. The load is further reduced as the NFV node only needs to handle the SYN and ACK packets, not the data traffic. Furthermore, the NFV nodes do not need to communicate with each other. Even so, the work in [44] has shown that NFV platform is capable of achieving production-level performance for carrier-grade NAT operations, a functionality used by T-SAC.

Based on the discussion on incremental deployment and flexible placement, there is no need to have a full deployment of the NFV nodes or the SDN switches upfront. Starting from a single programmable switch and NFV node, T-SAC can start diverting traffic within its capacity, only handling the amount of traffic it can handle. The rest of the traffic is handled by native anycasting.

Furthermore, T-SAC is capable of diverting traffic between potentially any servers. Based on the discussion on server groups in Section III-A1, there can be multiple levels of server groups, for instance, three levels – an intra-PoP server group, a nation-wide inter-PoP server group and a global inter-PoP server group. T-SAC may first start redirecting traffic within the intra-PoP server group, then the nation-wide server group, and finally the global server group. This layering approach of server groups not only improves the scalability of the system but also user experience as T-SAC will first attempt to divert traffic to a server close to the user, and then gradually move on to other servers if needed. In comparison, the layering approach in FastRoute [4] is also designed for scalability, but, it does not have the level of flexibility that T-SAC has.

VII. Related Work

Diverting traffic from an anycast IP address to a unicast IP address [7]–[9] or another anycast address [4] has been commonly used for creating scalable systems. We are particularly interested in FastRoute [4], as it aims to create a scalable anycast routing architecture for CDNs, and is currently deployed in production.

A. FastRoute

FastRoute [4] is a scalable anycast routing architecture developed by Microsoft for the Bing CDN to redirect traffic from overloaded servers. In FastRoute CDN servers are grouped into three layers (Fig. 17), namely, edge servers, regional hubs and the central hub from outer layer to inner layer. Each layer has a group of CDN servers and its anycast address. Each CDN server is collocated with a load-aware DNS server. When a server is overloaded, upon a DNS request new flows are redirected to its neighbouring inner layer. FastRoute and T-SAC are similar in that only new flows are redirected – existing flows stay with the current server without being interrupted.

However, T-SAC addresses a number of issues that FastRoute has.

1. To avoid the CDN servers redirecting to each other (see Fig. 11), FastRoute only allows redirection from an outer layer to its neighbouring inner layer, that is, from an edge server to a regional hub or a regional hub to the central hub, not the other way around or between the servers within the same layer. This results in inefficient use of server capacity. By adding a no-redirect flag, T-SAC allows redirection potentially between any servers without having the problem of the servers redirecting to each other.
2. FastRoute relies on a greedy heuristics for the redirection [55]. The fraction of traffic being redirected is proportional to $current\ load - overload\ threshold$. This means that FastRoute may face a dilemma: redirect not fast enough in the presence of sudden bursts of traffic, such as Flash Crowds [51], or unnecessarily redirect too much traffic. In T-SAC, traffic is redirected in a proactive, dynamic and smooth manner based on server dynamics.
3. FastRoute relies on load-aware DNS servers to redirect, and thus inherits the DNS-based problems (see Section II-A) [20], [41].

4. The redirection granularity of FastRoute is at LDNS level, that is, all the new flows from the users behind the LDNS will be redirected. Ours is at flow level.
5. FastRoute relies on the self-correlation between the cache server and its collocated DNS server, that is, DNS queries and subsequent user traffic to cache servers need to land on the same FastRoute node. This correlation does not always hold.
6. FastRoute only uses anycast addresses, giving CDN/ISP operators no control on which server the users are connected to. It also inherits the route flapping problem of anycast.
7. FastRoute does not support incremental deployment. If there was only one FastRoute node, all the anycast traffic to the CDN would have to be handled by this single node.

B. Other Studies

The work in [8] proposes a scalable CDN architecture using push-based notification. When a user request is sent to the anycast address of the CDN servers, the system chooses a server and explicitly asks the user to use the unicast address of the selected server. The architecture requires modifications to the client side, and does not specifically address the issue of traffic redirection among the servers.

PIAS [7] is a global anycast overlay, aiming to be scalable and replace the native anycast. In general, PIAS is similar to Mobile IP [56]. IAP (Ingress Anycast Proxy) and JAP (Join Anycast Proxy) in PIAS resembles Home Agent (HA) and Foreign Agent (FA) respectively in Mobile IP. PIAS uses $IP\ address + port\ number$ as the address of an anycast group. When IAP receives a packet for an anycast group for the first time, it enquires a Rendezvous Anycast Proxy (RAP), which selects a JAP based on a set of selection criteria such as proximity and load. The JAP then selects the most suitable anycast server to handle the traffic. Both JAP and anycast servers use unicast IP addresses while IAP uses anycast addresses. The JAP is cached by the IAP for future use. A tunnel is established from the IAP to the JAP. The JAP decapsulates the packet and performs DNAT rewriting its destination from the anycast group address to the unicast address of the server. The role of RAP is similar to a DNS server. In the simplest form, a single PIAS proxy can act as RAP, IAP and JAP. If there was only one such an overlay proxy all anycast traffic would have to be handled by this single proxy. For this reason, tens of thousands of PIAS proxies have to be deployed at once to make it work.

In contrast T-SAC can coexist with the native anycast. The NFV Redirection Node is not an overlay proxy. If there was only one such a node, the NFV node would only handle the traffic ISP/CDN providers want it to handle. All the other traffic would be handled by native anycasting. This means that T-SAC supports incremental deployment of the NFV Redirection Nodes. Another key difference is that in PIAS the redirection from one server to another will break existing TCP sessions. If the serving JAP is overloaded, the RAP identifies a new JAP and then all existing flows have to be re-established with the new JAP. Moreover, PIAS is not capable of flow-level redirection as port numbers are part of anycast addresses.

According to a blog article on CloudFlare's architecture [3], load balancing with TCP flows gets trickier (in comparison to UDP), as session states have to be maintained. There is no much detail provided. We suspect that the trickiness is about the scenarios that load balancing with TCP flows often results in the TCP sessions being broken, or maintaining TCP session states is not a trivial task [42]. While there are some load balancing solutions adopted by CDN operators, unfortunately these efforts are not well documented in public literature. The most relevant and well documented research is FastRoute [4]. Nevertheless, our architecture is focused on diverting traffic in a proactive, dynamic, smooth, non-interruptive and flexible manner.

MIMA, an architecture proposed in [9] is able to divert traffic among the servers in anycast-based CDNs. While MIMA provides a promising platform, its redirection capability is rather limited. MIMA can only redirect at a fixed ratio. In the event of high flow arrival rate the redirection may not be able to offload quickly enough. In the event of low flow arrival rate, MIMA may unnecessarily offload traffic to other servers. Combined with the hysteresis-based redirection in MIMA, the amount of traffic handled by the current server may rapidly fluctuate between the hysteresis limits, making it difficult to achieve an equilibrium load level (see Fig. 9). In contrast, T-SAC redirects traffic in a proactive, dynamic and smooth manner. The redirection only starts when the server is at the risk of being overloaded. The Redirection Ratio is adjusted based on server dynamics. This ensures that only the right amount of traffic is redirected without unnecessarily offloading to other servers. MIMA has a mechanism similar to the no-redirect flag in our architecture. However, it does not address the flapping problem of the flag, adding unnecessary overhead to the system.

SoDA (Software Defined Anycast) proposed in [30] enables anycast routing based on a thin layer of SDN functionality. SoDA balances the load of CDN proxy clusters by moving load balancing functionality into the ISP network. In SoDA, the SDN-based redirection at the edge router rewrites the anycast IP address of a proxy cluster into the anycast IP address of another proxy cluster. The traffic in both directions (client to server and server to client) needs to go through the edge router for the NAT operations, that is, DNAT and SNAT, respectively. The NAT operations may break long-lived TCP sessions if redirecting to a different proxy clusters happens before the TCP sessions complete. However, it shows that the amount of traffic affected by session losses is extremely low. Although it is a simulation based observation, it is backed up by real-world measurement shown in [31]. It is also worth noting that if the granularity of the load balancing increases the amount of traffic being affected by session losses will increase. Similarly, our initial work in [41] also adopted DNAT and SNAT operations at an edge router for redirecting traffic in DNS-based CDNs.

ISP and CDN operators have been collaborating, and the ISP-CDN collaboration problem has been well studied [40], [42], [57]–[59]. Moreover, the IETF ALTO working group [60] has proposed the Application-Layer Traffic Optimization (ALTO) protocol, which enables network operators to support application-level traffic engineering [61], [62]. This includes providing an interface between ISPs and CDNs to facilitate ISP-CDN collaboration. More recently, content providers tend to employ multiple CDNs to serve their content. This allows the switching between CDNs, that is, the serving CDN can be selected based on costs, performance or availability. This is essentially the concept of Meta-CDNs, also known as CDN broker or Multi-CDN selectors [12], [63]. Meta-CDN is an infrastructure that enables routing between CDNs with the routing logic provided by content providers. Cedexis [64], a prominent Meta-CDN, works primarily the same way as a DNS-based CDN, redirecting the requesting user to the CDN selected for content delivery [12]. The selected serving CDN will then follow its own server selection mechanism and identify the suitable cache server. Our architecture is designed to route traffic within a CDN, thus complementary to Meta-CDN. An extension of our architecture, however, can be used to realize Meta-CDN. The work in [12] presents a broad assessment on the operation of Meta-CDN in the wild.

VIII. Conclusions

In this paper we proposed T-SAC, a scalable anycast-based CDN architecture that enables precise and fine-grained control on how traffic is routed. T-SAC leverages SDN, NFV, as well as the combination of unicast and anycast for traffic redirection among the cache servers.

We designed a set of mechanisms including a load-based traffic redirection method and a 1-bit no-redirect flag, to ensure that the traffic redirection is performed in a proactive, flexible, dynamic, smooth and non-interruptive manner. Redirection is activated when the server is at the risk of being overloaded. Only the right amount of traffic is redirected to maintain the performance of the current server without unnecessarily offloading to other servers. Traffic can be potentially redirected to any servers that are suitable to handle additional traffic. This is achieved by using a single-bit no-redirect flag. Only new flows are redirected. The existing flows stay with the current server without being interrupted. The anycast route flapping problem is mitigated because of the use of the server's unicast IP address. The redirection is completely transparent to the users. The proposed architecture only requires a small amount of state information: the load of the server to be offloaded and the no-redirect flag of the servers that are the candidates to receive the redirected traffic.

Capitalizing on the programmability of SDN/NFV and the flexibility of virtualization, T-SAC requires very few changes to existing ISP/CDN networks: an NFV Redirection Node and an SDN switch that can forward anycast traffic to the NFV node. The SDN switch is offloaded by the NFV node, resulting in a simple and scalable data plane. By manipulating the forwarding rules in the switch, the load of the NFV node is manageable. No communications are needed between the NFV nodes and the nodes only handle the TCP SYN and ACK packets. Thus, the scalability issues of the NFV node are controllable. The nature of NFV makes it convenient to spin up an NFV node and place it strategically in the network. The architecture can coexist with native anycasting and thus supports incremental deployment. T-SAC is simple yet scalable and effective, suitable for practical deployment.

References

[1] *Cisco Visual Networking Index: Forecast And Methodology, 2016–2021*. Accessed: Sep. 25, 2018. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html

[2] M. Prince. *A Brief Primer on Anycast*. Accessed: Sep. 25, 2018. [Online]. Available: https://blog.cloudflare.com/a-brief-anycast-primer/

[3] *Load Balancing Without Load Balancers*. Accessed: Sep. 25, 2018. [Online]. Available: https://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/

[4] A. Flavel *et al.*, "FastRoute: A scalable load-aware anycast routing architecture for modern CDNs," in *Proc. USENIX NSDI*, 2015, pp. 381–394.

[5] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the performance of an anycast CDN," in *Proc. ACM IMC*, 2015, pp. 531–537.

[6] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, Aug. 2010.

[7] H. Ballani and P. Francis, "Towards a global IP anycast service," in *Proc. ACM SIGCOMM*, 2005, pp. 301–312.

[8] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van Der Merwe, "A practical architecture for an anycast CDN," *ACM Trans. Web*, vol. 5, no. 4, pp. 17:1–17:29, Oct. 2011.

[9] J. Lai and Q. Fu, "Man-in-the-middle anycast (MIMA): CDN user-server assignment becomes flexible," in *Proc. LCN*, 2016, pp. 451–459.

[10] ETSI. *Network Functions Virtualisation*. Accessed: Sep. 25, 2018. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[11] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[12] O. Hohlfeld, J. Rüth, K. Wolsing, and T. Zimmermann, "Characterizing a meta-CDN," in *Proc. PAM*, 2018, pp. 114–128.

[13] *RYU SDN Framework*. Accessed: Sep. 25, 2018. [Online]. Available: https://osrg.github.io/ryu/

[14] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comp. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[15] *Mininet Overview*. Accessed: Sep. 25, 2018. [Online]. Available: http://mininet.org/overview/

[16] *Amazon Web Services*. Accessed: Sep. 25, 2018. [Online]. Available: https://aws.amazon.com/

[17] *Node.JS*. Accessed: Sep. 25, 2018. [Online]. Available: https://nodejs.org/

[18] H. Niki. *GNU Wget*. Accessed: Sep. 25, 2018. [Online]. Available: https://www.gnu.org/software/wget/

[19] *Dash Industry Forum*. Accessed: Sep. 25, 2018. [Online]. Available: http://dashif.org/

[20] J. Lai, Q. Fu, and T. Moors, "Using SDN and NFV to enhance request rerouting in ISP-CDN collaborations," *Comput. Netw.*, vol. 113, pp. 176–187, Feb. 2017.

[21] A. Qureshi, R. Weber, H. Balakrishnan, J. Guttag, and B. Maggs, "Cutting the electric bill for Internet-scale systems," in *Proc. ACM SIGCOMM*, 2009, pp. 123–134.

[22] H. H. Liu, Y. Wang, Y. R. Yang, H. Wang, and C. Tian, "Optimizing cost and performance for content multihoming," in *Proc. ACM SIGCOMM*, 2012, pp. 371–382.

[23] F. Chen, R. K. Sitaraman, and M. Torres, "End-user mapping: Next generation request routing for content delivery," in *Proc. ACM SIGCOMM*, 2015, pp. 167–181.

[24] C. Contavalli, W. van der Gaast, D. Lawrence, and W. Kumari, *Client Subnet in DNS Queries*, document RFC 7871, May 2016.

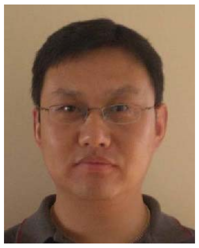
[25] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 3, pp. 52–66, Jul. 2015.

[26] *Cloudflare*. Accessed: Sep. 25, 2018. [Online]. Available: https://www.cloudflare.com

[27] *Amazon*. Accessed: Sep. 25, 2018. [Online]. Available: https://aws.amazon.com/route53/faqs/

[28] *Google Cloud Platform*. Accessed: Sep. 25, 2018. [Online]. Available: https://cloud.google.com/load-balancing/

[29] D. Giordano, D. Cicalese, A. Finamore, M. M. M. Mellia, D. Z. Joumblatt, and D. Rossi, "A first characterization of anycast traffic from passive traces," in *Proc. IFIP TMA*, 2016, pp. 1–8.
[30] M. Wichtlhuber *et al.*, "SoDA: Enabling CDN-ISP collaboration with software defined anycast," in *Proc. IFIP Netw.*, 2017, pp. 1–9.
[31] M. Levine, B. Lyon, and T. Underwood. (2006). *TCP Anycast—Don't Believe the FUD. Operational Experience With TCP and Anycast*. Accessed: Sep. 25, 2018. [Online]. Available: https://www.nanog.org/meetings/nanog37/presentations/matt.levine.pdf
[32] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, 2013, pp. 3–14.
[33] *Cord Project*. Accessed: Sep. 25, 2018. [Online]. Available: https://opencord.org/
[34] D. Cicalese, D. Z. Joumblatt, D. Rossi, M.-O. Buob, J. Augé, and T. Friedman, "Latency-based anycast geolocation: Algorithms, software, and data sets," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 6, pp. 1889–1903, Jun. 2016.
[35] M. T. Arashloo, Y. Koral, M. Greenberg, J. Rexford, and D. Walker, "SNAP: Stateful network-wide abstractions for packet processing," in *Proc. ACM SIGCOMM*, 2016, pp. 29–43.
[36] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "VeriFlow: Verifying network-wide invariants in real time," in *Proc. USENIX NSDI*, 2013, pp. 49–54.
[37] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012, pp. 13–24.
[38] H. J. Stuart, "The exponentially weighted moving average," *J. Qual. Technol.*, vol. 18, no. 4, pp. 203–210, 1986.
[39] K. Cho, H. Jung, M. Lee, D. Ko, T. Kwon, and Y. Choi, "How can an isp merge with a CDN?" *IEEE Commun. Mag.*, vol. 49, no. 10, pp. 156–162, Dec. 2011.
[40] B. Frank *et al.*, "Pushing CDN-ISP collaboration to the limit," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 2, pp. 34–44, 2013.
[41] J. Lai, Q. Fu, and T. Moors, "Rapid IP rerouting with sdn and NFV," in *Proc. IEEE Globecom*, Dec. 2015, pp. 1–7.
[42] M. Wichtlhuber, R. Reinecke, and D. Hausheer, "An SDN-based CDN/ISP collaboration architecture for managing high-volume flows," *IEEE Trans. Netw. Service Manag.* vol. 12, no. 1, pp. 48–60, Mar. 2015.
[43] A. K. F. Ahmed, Z. Shafiq, and A. X. Liu, "Optimizing Internet transit routing for content delivery networks," in *Proc. IEEE ICNP*, Nov. 2016, pp. 1–10.
[44] J. Martins *et al.*, "Clickos and the art of network function virtualization," in *Proc. USENIX NSDI*, 2014, pp. 459–473.
[45] R. Stoenescu *et al.*, "In-net: In-network processing for the masses," in *Proc. EuroSys*, 2015, p. 23.
[46] H. Ballani *et al.*, "Enabling end-host network functions," in *Proc. ACM Conf. Special Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 493–507.
[47] *Big Buck Bunny*. Accessed: Sep. 25, 2018. [Online]. Available: http://dash.edgesuite.net/akamai/bbb_30fps
[48] *Dash.JS*. Accessed: Sep. 25, 2018. [Online]. Available: https://github.com/Dash-Industry-Forum/dash.js/
[49] *Chromium Project*. Accessed: Sep. 25, 2018. [Online]. Available: https://www.chromium.org/
[50] *REANNZ ISP Network*. Accessed: Sep. 25, 2018. [Online]. Available: https://reannz.co.nz/services/networking/network/
[51] P. Wendell and M. J. Freedman, "Going viral: Flash crowds in an open CDN," in *Proc. ACM IMC*, 2011, pp. 549–558.
[52] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang, "QDASH: A QoE-aware DASH system," in *Proc. ACM MMSys*, 2012, pp. 11–22.
[53] A. Bentaleb, A. C. Begen, and R. Zimmermann, "SDNDASH: Improving QoE of HTTP adaptive streaming using software defined networking," in *Proc. ACM MM*, 2016, pp. 1296–1305.
[54] *FastRoute NSDI Slides*. Accessed: Sep. 25, 2018. [Online]. Available: https://www.usenix.org/sites/default/files/conference/protected-files/nsdi15_slides_flavel.pdf
[55] A. Sinha, P. Mani, J. Liu, A. Flavel, and D. Maltz, "Distributed load management algorithms in anycast-based cdns," *Comput. Netw.*, vol. 115, pp. 1–15, Mar. 2017.
[56] C. E. Perkins, "Mobile IP," *IEEE Commun. Mag.*, vol. 35, no. 5, pp. 84–99, May 1997.
[57] I. Poese, B. Frank, G. Smaragdakis, S. Uhlig, A. Feldmann, and B. Maggs, "Enabling content-aware traffic engineering," *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 5, pp. 21–28, Sep. 2012.
[58] I. Poese, B. Frank, S. Knight, N. Semmler, and G. Smaragdakis, "PaDIS emulator: An emulator to evaluate CDN-ISP collaboration," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2012, pp. 81–82.
[59] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering in an isp network," in *Proc. 11th Int. Joint Conf. Meas. Modeling Comput. Syst. (SIGMETRICS)*, 2009, pp. 239–250.
[60] Information Assurance Working Group. *Application-Layer Traffic Optimization*. Accessed: Sep. 25, 2018. [Online]. Available: https://datatracker.ietf.org/wg/alto/about/
[61] R. Alimi *et al.*, *Application-Layer Traffic Optimization (ALTO) Protocol*, documrent RFC 7285, Sep. 2014.
[62] M. Stiemerling, S. Kiesel, M. Scharf, H. Seidel, and S. Previdi, *Application-Layer Traffic Optimization (ALTO) Deployment Considerations*, document RFC7971, Oct. 2016.
[63] M. K. Mukerjee, I. N. Bozkurt, D. Ray, B. M. Maggs, S. Seshan, and H. Zhang, "Redesigning cdn-broker interactions for improved content delivery," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2017.
[64] *Cedexis*. Accessed: Sep. 25, 2018. [Online]. Available: https://www.cedexis.com/



Qiang Fu received the Ph.D. degree in telecommunications engineering from The University of Queensland, Australia. He is currently a Senior Lecturer in network engineering with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include Internet architecture and protocols, wireless and mobile systems, and network measurement and security.

Bradley Rutter received the B.E. degree (Hons.) in network engineering from the Victoria University of Wellington in 2017. He is currently a Network Engineer with Datacom, Wellington, New Zealand.

Hao Li (S'13–M'16) received the B.S. and Ph.D. degrees in computer science from Xi'an Jiaotong University in 2010 and 2016, respectively. He is currently an Assistant Professor with the Department of Computer Science and Technology, Xi'an Jiaotong University. His main research interests include computer networking systems, network measurement and monitoring, and software-defined networking.

Peng Zhang received the Ph.D. degree in computer science from Tsinghua University in 2013. He is currently an Associate Professor with the Department of Computer Science and Technology, Xi'an Jiaotong University, China. He has been a Visiting Student at The Chinese University of Hong Kong and Yale University. His research interests include network security and software-defined networking.

Chengchen Hu received the Ph.D. degree from Tsinghua University, Beijing, China, in 2008. From 2008 to 2010, he was an Assistant Research Professor with Tsinghua University. Later, he joined the Department of Computer Science and Technology, Xi'an Jiaotong University (XJTU), Xi'an, China, where he served as an Associate Professor first and a Professor since 2016 and the Department Head from 2016 to 2017. Since 2017, he has been on leave from XJTU and became the Principal Engineer and Director leading Xilinx Research Labs Asia Pacific, Singapore.

His main research interests include network measurement, cloud data center networking, and software-defined networking. He severed in the organization committee and technical program committee of several conferences, e.g., INFOCOM, IWQoS, GLOBECOM, ICC, ANCS, and Networking. He was a recipient of a fellowship from the European Research Consortium for Informatics and Mathematics, Microsoft "Star-Track" Young Faculty Program, New Century Excellent Talents in University awarded by the Ministry of Education, China.

Tian Pan (S'10–M'14) received the B.S. degree from Northwestern Polytechnical University, Xi'an, China, in 2009, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2015. He was a Post-Doctoral Researcher with the Beijing University of Posts and Telecommunications from 2015 to 2017, where he has been an Assistant Professor since 2017. His research interests include router architecture, software-defined networking, programmable data plane, and machine learning for network applications.

Zhangqin Huang received the B.E., M.S., and Ph.D. degrees in computer science from Xi'an Jiaotong University in 1986, 1989, and 2000, respectively. From 1989 to 2001, he was a Lecturer and an Associate Professor at Xi'an Jiaotong University. From 2001 to 2003, he carried out post-doctoral research at the Eindhoven University of Technology, The Netherlands. Since 2003, he has been a Professor with the Beijing University of Technology, where he currently serves as an Executive Vice Director of the Embedded Software System Institute and the Head of the Embedded System Department, National Model Software College. He is also a Professor at the Beijing University of Technology. His research interests include system hardware and software co-design, wireless communication protocols, SoC, IoT, and Internet architectures.

Yibin Hou received the Ph.D. degree in computer science from the Eindhoven University of Technology, The Netherlands, in 1986. He is currently a Professor with the School of Software Engineering, Beijing University of Technology, and the Director of the Beijing Research Institute for Internet of Things, Beijing, China. He was a Professor with the Department of Computer Science, Xian Jiaotong University, from 1991 to 2002, and the Dean of the School of Software Engineering, Beijing University of Technology, from 2002 to 2016. His research interests include software engineering, embedded systems, and Internet of Things.