# MOSAIC: Stateless Mobility for HTTP-based Applications

Wonsang Song*, Georg Hampel†, Anil Rana†, Thierry Klein† and Henning Schulzrinne*
*Columbia University
{wonsang, hgs}@cs.columbia.edu
†Bell Labs
{georg.hampel, anil.rana, thierry.klein}@alcatel-lucent.com

***Abstract*—We present a mobility solution for stateless applications, where the mobile host can change its IP address as well as the content servers used by ongoing client sessions. This allows content retrieval to always use the locally optimal source when the host moves between networks. We refer to this approach as "stateless mobility" since neither the network nor the content servers hold mobility-related state information.**

**Our mobility solution, referred to as MOSAIC, is applied to HTTP sessions using GET method invocation, which represent a large fraction of mobile Internet traffic. By moving MOSAIC underneath the socket interface, we create a generic application-independent feature. We realized a lightweight implementation using an L3 packet filter, which promises easy portability to other platforms. MOSAIC's overall concept is evaluated via measurements using public Internet services.**

***Index Terms*—stateless mobility, vertical handover, multi-homing, HTTP content retrieval**

## I. Introduction

Content retrieval by stateless client-server applications generates a major portion of today's Internet traffic [1]. Examples of such applications are web browsing, video streaming, and file transfer. These applications are referred to as "stateless" since the server remains agnostic to the client-specific session state [2]. While stateless applications are not confined to a particular protocol, the World Wide Web has established HTTP with GET method invocation as the de facto standard for these applications [1].

Stateless applications have the advantage that contents can be replicated and conveniently cached on Content Distribution Networks (CDNs) closer to the edge [3]. Moreover, different parts of a single content can be retrieved from different sources. This allows a client to resume a download where it left off after connection failure. For this purpose, HTTP 1.1 supports the *partial* GET method, which permits the client to confine the request to those data that are still missing [4].

We propose to extend the inherent advantages of stateless applications to support vertical session mobility [5]. Vertical session mobility refers to the scenarios where hosts migrate their ongoing sessions to new IP addresses. The need for vertical mobility has surged with the introduction of smartphones and tablets, whose multi-homing capabilities provide access to a multitude of networks with overlapping coverage areas. While most of these devices autonomously manage network authentication, vertical session mobility is usually not supported [6].

In analogy to the concept of stateless applications, we introduce the concept of *stateless mobility* where only the mobile host holds mobility-related state information. Stateless mobility permits the host not only to migrate a data session to a new IP address but also to switch the content server used by this session. Consequently, the optimal server can be selected at each mobility event, which upholds the spirit of content locality across mobility events.

Stateless mobility stands in stark contrast to the present *stateful mobility* protocols, such as Mobile IP [7], which tether the client session to a specific content source.

We have developed a vertical mobility solution for HTTP-based applications, referred to as MOSAIC, which is stateless in nature and transparent to client applications. Application transparency is a major requirement for such a solution since it provides backward compatibility with existing applications and eliminates the need to furnish every future application with its own mobility feature.

Application transparency implies that MOSAIC's implementation has to occur *below* the socket interface. Any implementation of that kind violates the logical protocol layering scheme, which assigns HTTP to the OSI L5-L7 space *above* the socket interface. Application transparency therefore has to be seen as a concession to practicality rather than a fundamental requirement for HTTP-based mobility.

We implemented MOSAIC on top of an L3 packet filter in Linux. We argue that such an implementation is not only lightweight, but also easily portable to other operating systems. Using this implementation, we have conducted a variety of trials on actual websites including web browsing, video streaming, and downlink file transfer to prove the concept of stateless mobility and to explore its limitations.

The following section describes the MOSAIC solution. In Section III, this solution is compared to related work. Section IV discusses implementation tradeoffs for MOSAIC and emphasizes on our packet filter realization. Section V shows measurement data conducted with the MOSAIC implementation. The results of these trials as well as limitations to stateless mobility are discussed in Section VI. We summarize and outline future work in the conclusion.

978-1-4673-1430-5/12/$31.00 ©2012 IEEE

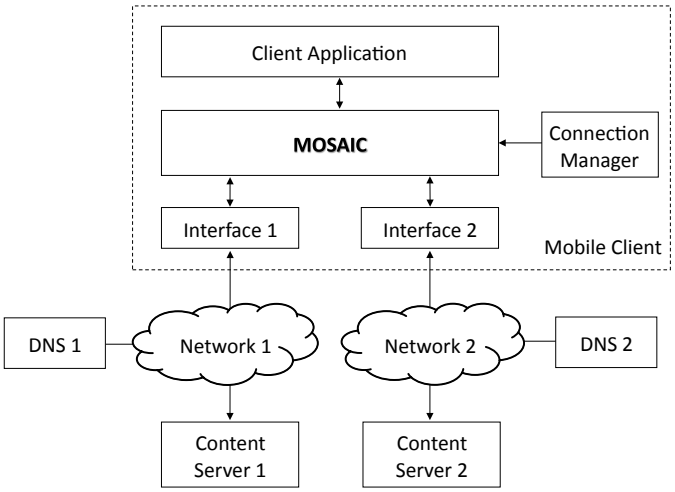



Fig. 1. MOSAIC architecture.

## II. MOSAIC Solution

MOSAIC introduces the concept of "stateless mobility" for mobile hosts that run client sessions pertaining to stateless applications. Stateless mobility allows the mobile host to change its IP address as well as the content sources used by the ongoing client sessions. Since the new IP address can belong to the same or to a different interface, stateless mobility also supports multi-homing.

Stateless mobility confines all mobility-related operations to the mobile host. Hence, neither the content server nor the network has to hold any mobility-related state information, which avoids the need for a separate mobility protocol. Instead, a new content retrieval request is sent after each mobility event, which holds information on the remaining fraction of content. In case content retrieval is based on HTTP 1.1, the partial GET method supports the `Range` header for this purpose, which holds the start and end byte index of the remaining content fragment.

Since HTTP has become the de facto standard for stateless applications, we consider it sufficient to restrict stateless mobility to HTTP for the moment.

Figure 1 depicts the architecture of MOSAIC for a multi-homed host with simultaneous access to two different networks. This architecture reflects a situation encountered by a smartphone, tablet, or laptop that enjoys connectivity to a 3G/4G cellular network as well as to an Internet service provider via Wi-Fi. In this architecture, MOSAIC resides between client applications and network interfaces. We assume that content is replicated and hosted by a separate content server in each access network. Further, each network supports its own DNS service. The client selects the locally optimal content server using the DNS-based server selection scheme [8].

The host is assumed to also hold a connection manager, which monitors interface availability and makes the decision on when a handover should occur. In such an event, it sends a trigger message to MOSAIC, which executes the session migration request.

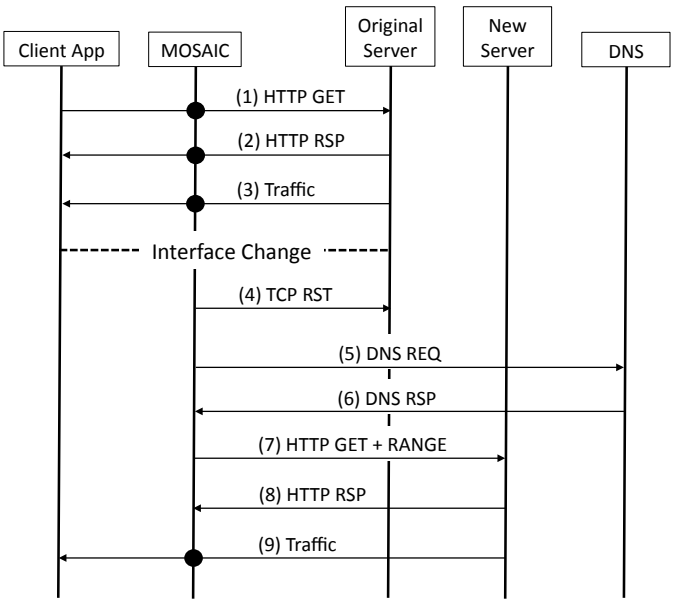



Fig. 2. MOSAIC operation.

Figure 2 shows MOSAIC's operation. To start the session, the client resolves the content server's IP address via a DNS request, establishes a TCP connection, and sends an HTTP GET request to the server (1).

Since MOSAIC resides between client applications and network interfaces, it inspects the HTTP request and decides if mobility support is appropriate. In such a case, MOSAIC caches the HTTP GET request and inspects all following data received on this connection. The content server returns an HTTP response (2), and then it starts streaming traffic data (3). MOSAIC analyzes the passing HTTP response for reasons discussed below, and it counts the bytes of content the client has received.

When an interface change occurs, MOSAIC terminates the current TCP connection (4). Then it initiates a DNS handshake on the new network to find the locally optimal content server. For this purpose, it inserts into the DNS query the host part of the HTTP URL contained in the cached HTTP GET request (5). After obtaining the DNS response (6), MOSAIC establishes a TCP connection with the new server, where it sends a partial HTTP GET request. This request contains the initial HTTP GET headers and a `Range` header (7), whose start value is based on the total byte count of data received.

From the data stream sent by the new server, MOSAIC extracts and suppresses the HTTP response (8) and relays all content information to the client process (9). MOSAIC continues to maintain the byte count of the content being retrieved, in case another interface change occurs.

The session migration shown in Figure 2 can also be applied to a mobility event where the same physical interface is moved from one network to another.

Since MOSAIC uses independent TCP connections and HTTP requests on each network, it is compliant with middle

boxes such as firewalls, network address translators, intrusion detection systems, and HTTP proxies. In the presence of HTTP proxies, the proxy address represents the IP address of the local content server and a separate DNS request is not required.

MOSAIC also supports HTTP redirections. HTTP redirection can be used to direct clients to the optimal CDN node in addition to the DNS-based scheme [9]. YouTube, for example, uses both the DNS-based and HTTP redirection mechanism for their video delivery [10]. In such cases, the initial HTTP response provides an alternative URL, where the content should be retrieved. Upon reception of the redirection header, the client restarts the content retrieval session using this alternative URL. Since the redirected content retrieval process follows the same sequence as shown in Figure 2, it automatically enjoys MOSAIC support.

One problem may occur when the alternative URL provided in the redirection header is local to the network where it was sent. In this case, MOSAIC's DNS request for this URL will fail on any other network. To handle this scenario, MOSAIC inspects the HTTP response for redirection headers and caches the corresponding URLs together with the global URL of the initial request. From this, MOSAIC can recognize client sessions using redirected URLs. Consequently, MOSAIC uses the global URL instead of the redirected URL for DNS requests on a new network.

MOSAIC's session migration is conducted in a break-before-make manner even when the old and the new interface are available simultaneously. This break is due to the fact that a firm byte count has to be established on the old interface before a partial HTTP GET request can be sent on the new interface. The delay can be reduced to one round-trip time if DNS handshake and connection establishment on the new network are conducted before the break. More refined methods using estimates on future byte counts may reduce this gap even further. Our measurements using typical web applications, however, indicate that the break is rarely noticeable.

## III. Related Work

There are a vast number of mobility, multi-homing, and multipath solutions available for OSI protocol layers 3 to 7 [11], [12]. Compared to MOSAIC's state*less* mobility approach, all these solutions are inherently state*ful* since they rely on a protocol that exchanges mobility-related state information between two end points acting on behalf of mobile host and correspondent. These end points can reside on network side, such as Mobility Access Gateway (MAG) and Local Mobility Anchor (LMA) in Proxy Mobile IP [13], or they can be embedded into the end hosts as proposed by the Host Identifier Protocol (HIP) [14], E2E Host Mobility [15], and Multipath TCP [12].

Stateful and stateless mobility define a tradeoff: While stateful mobility requires two end points to exchange state information via a protocol, it supports both stateful and stateless applications. Stateless mobility, in contrast, can only support stateless applications, but it retains content locality and remains confined to the client's host.

Horizontal handovers between neighboring cells are local in nature and best served with state*ful* mobility. State*less* mobility would perform poorly due to the delay associated with frequent connection re-establishment and consecutive TCP slow start. Further, content locality is retained implicitly since mobility is local.

For vertical handovers, however, stateless applications should use stateless mobility due to its inherent advantages.

The underlying principles of stateless mobility have also been discussed within the framework of Information-Centric Networking (ICN) [16]. ICN advocates a new networking paradigm based on named contents. ICN is not yet widely adopted on the Internet.

Also some Dynamic Adaptive Streaming over HTTP (DASH) applications support vertical mobility in compliance with the stateless mobility concept [17]. Since DASH conducts content retrieval via a sequence of requests for small data chunks, handovers can be neatly inserted between the delivery of one data chunk and the request for the next. While this form of mobility could also be provided in application-transparent manner underneath the socket layer, it would be restricted to DASH applications. MOSAIC, however, can support both conventional HTTP applications as well as DASH applications.

## IV. MOSAIC Implementation

Our goal is to provide vertical mobility to all stateless HTTP applications through MOSAIC. In order to support web browsers, we can implement MOSAIC as a plug-in or as a function in HTTP proxies. However, mobile computing has brought a new breed of applications that rely on HTTP content retrieval. With the arrival of Apple App Store and Android Market, mobile applications have created their own ecosystem, demanding easy, swift, and competitive code development. In such an environment, it is desirable to provide mobility support as a generic application-independent feature, so that all applications–whether they are built with mobility in mind or not–can benefit from the mobility support. This requirement poses new challenges.

While MOSAIC could be supported on session layer for this purpose, there is no generally accepted interface from application to session layer. In fact, most application development feels bound by the socket interface only. Therefore, MOSAIC needs to be implemented below the socket interface to enjoy broad application-layer transparency. Such a cross-layer feature relocation may appear awkward, but it is commonly applied, for instance, by flow-based mobility proposals that weave L4 semantics into an L3 solution [18].

Underneath the socket interface, MOSAIC can be integrated into the transport layer or into a packet filter running on network layer. We chose the latter approach since it provides better code portability to other platforms. Most modern operating systems already support hooks for packet filtering and the interface is generally straightforward.

Further, Linux–our platform of choice–provides a framework for packet filtering called Netfilter [19], which allows packet modifications to be conducted in user space. The detour

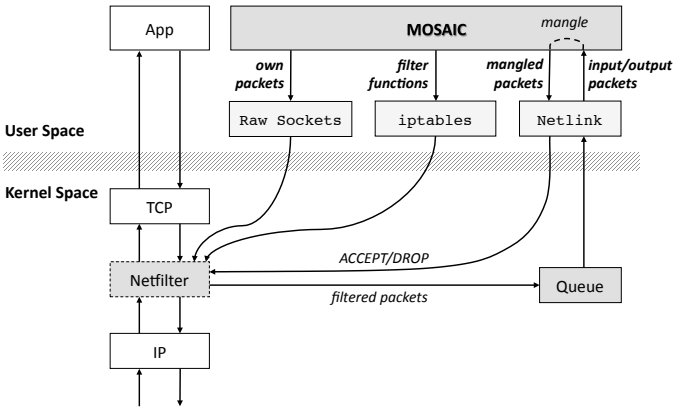



Fig. 3. MOSAIC Linux implementation.

through user space permits fast prototyping, debugging, and trialing. Afterwards, the mature code can be provided as a kernel module.

Figure 3 describes our Linux implementation. Netfilter includes iptables, a command to inject packet filtering rules into the kernel. The rules can specify which packets are sent to a queue, e.g., for user-space modifications. MOSAIC sets the packet selection rules through iptables and picks up the selected packets from the Netfilter queue via a Netlink socket [20]. MOSAIC then inspects and eventually modifies the packet content (discussed below) before it re-inserts the packet into the traffic stream. The packet can also be discarded. MOSAIC further supports a Raw Sockets API to create its own packets.

When no content retrieval application is running, MOSAIC only selects outgoing TCP SYN packets for inspection, which minimizes MOSAIC's processing overhead. When a SYN packet arrives, MOSAIC extends the packet selection rules to include all packets pertaining to this TCP connection and follows the TCP setup handshake via packet inspection.

When the connection is established, MOSAIC searches the first payload packet for the HTTP GET request. If such a request is not contained or if MOSAIC decides to omit mobility support for the URL contained in this request, it discontinues packet monitoring for this connection. Otherwise, it keeps track of both TCP and HTTP state information derived from the passing packets.

The TCP state information includes the connection's 4-tuple consisting of source and destination IP addresses and port numbers as well as the most recent TCP sequence (SEQ) and TCP acknowledgement (ACK) numbers.

The HTTP state information includes the HTTP URL and the aggregate byte number of content received so far. This byte number is based on the server's initial SEQ number, the aggregate ACK number of the last outgoing acknowledgement, and the length of the server's HTTP response, which has to be discounted. The aggregate ACK number only reflects data that have arrived in sequence, since only those data are delivered to the client application. MOSAIC also stores a copy of the HTTP headers in the GET request.

When an interface change is due for this connection, MOSAIC creates a TCP RST packet using the cached TCP state information and transmits this packet on the raw socket. Then, it performs a DNS lookup as discussed in Section II, which determines the new content server's IP address. The new IP address of the mobile host is provided by the connection manager, the local port number is selected randomly, and the remote port number remains the same as before. MOSAIC instructs Netfilter to select incoming TCP packets with this new 4-tuple and to ignore all further TCP packets that arrive with the old 4-tuple.

Then, MOSAIC initiates a TCP connection setup with the new content server by creating a SYN packet with the new 4-tuple and sending it on the raw socket. The initial SEQ number on this packet is created via a random process. Upon arrival of the SYN-ACK return packet, MOSAIC creates and transmits an appropriate ACK response, while the SYN-ACK packet is discarded. In this manner, a TCP control block is created on the new content server but not on the mobile host. Instead, all necessary TCP state information is cached by MOSAIC.

After sending the ACK packet, MOSAIC creates a payload packet, which contains a partial GET request consistent of the initial HTTP GET request and the `Range` header. The `Range` header holds in the start field the index of the next byte of content expected and in the end field the total byte count obtained from the initial HTTP response. MOSAIC sends this packet to the new server and discards the server's HTTP response. If the HTTP response packet also contains content, MOSAIC eliminates the HTTP response header and shifts the content up to the packet header.

On this and all following packets, MOSAIC updates the fields for IP addresses, local port number, TCP SEQ and ACK numbers in a manner that matches the old TCP end point on the mobile host to the new TCP end point on the new content server. The packets checksums have to be updated accordingly. After these modifications, MOSAIC injects the packets back to the kernel. MOSAIC continues counting bytes of the content in case another mobility event occurs.

Our implementation has very low upfront cost, since packet modifications only occur after the first mobility event. This permits mobility support to be provided opportunistically, i.e., even if the application's duration and mobility requirements are not known a priori. The measurements presented in the next section confirm MOSAIC's small processing overhead.

## V. Mobility Trials with MOSAIC

We evaluated MOSAIC for three dominant types of HTTP traffic namely web browsing, downlink file transfer, and video streaming. These trials were conducted using actual websites.

MOSAIC was run on a laptop equipped with one Wi-Fi interface (802.11g) and one Ethernet interface (1 GB/s). The trial setup resembled that of Figure 1, where both network interfaces were connected to the Internet via independent subnets. The laptop was a Lenovo T61 with a 2.1 GHz Intel Core 2 Duo processor, which ran Ubuntu 11.04 on top of a Linux 2.6.38 kernel.

TABLE I
FILE DOWNLOADING WEBSITES AND MOSAIC COMPLIANCE.

| Name (URL) | File size | MOSAIC compliance |
|---|---|---|
| Ubuntu Linux ISO image (*http://mirror.anl.gov/pub/ubuntu-iso/DVDs-Ubuntu/11.10/release/ubuntu-11.10-dvd-i386.iso*) | 1.5 GB | Yes |
| Java SE JDK (*http://download.oracle.com/otn-pub/java/jdk/7u2-b13/jdk-7u2-linux-i586.rpm*) | 63.6 MB | Yes |
| Adobe Flash Player (*http://fpdownload.macromedia.com/pub/flashplayer/updaters/11/flashplayer_11_ax_debug_32bit.exe*) | 4.5 MB | Yes |

TABLE II
HTTP VIDEO STREAMING WEB SITES AND MOSAIC COMPATIBILITY.

| Name (URL) | Technology | MOSAIC compliance |
|---|---|---|
| YouTube (*http://www.youtube.com/watch?v=plxNfU-PA2c*) | Progressive (Flash and HTML5) | Yes |
| TED (*http://video.ted.com/talk/stream/2011U/None/MattCutts_2011U-950k.mp4*) | Progressive (HTML5) | Yes |
| Akamai HD Network Demo (*http://wwwns.akamai.com/hdnetwork/demo/flash/hds/index.html*) | Adobe HTTP Dynamic Streaming | Yes |
| MS Experience Smooth Streaming (*http://www.iis.net/media/experiencesmoothstreaming*) | MS Smooth Streaming | Yes |

For the HTTP client, we used the Google Chrome 13.0 web browser, which held plug-ins for Adobe Flash Player 10.3 and Moonlight 4.0 to support the video streaming trials (Moonlight is a Linux implementation of Microsoft Silverlight).

### A. MOSAIC Compliance

Any HTTP service is MOSAIC-compliant as long as it supports HTTP partial GET, which is an optional feature in HTTP 1.1. The first trials verified MOSAIC-compliance for a set of websites. For this purpose, an HTTP partial GET request was sent to each website, and the status code in the HTTP response was examined. If the status code was "206 Partial Content", partial GET was supported. Otherwise, the website responded with "200 OK". The start byte index in the `Range` header was set to a small value, i.e., 100, to make sure it never exceeded the total size of the content. For the websites supporting partial GET, we explicitly verified MOSAIC's proper operation by conducting handover trials.

We evaluated MOSAIC-compliance for web browsing using the top five websites according to Alexa.com. These websites were www.google.com, www.facebook.com, www.youtube.com, www.yahoo.com, and www.baidu.com. We found that none of these websites supported the partial GET feature. The lack of support for partial GET may be due to the small size of these webpages, which makes this feature rather unnecessary.

Downlink file transfer was evaluated for Ubuntu Linux ISO image, Oracle Java Development Kit (JDK), and Adobe Flash Player installer. The corresponding URLs and file sizes are shown in Table I. In contrast to the prior websites, all the selected file-transfer services did support partial GET.

Finally, we evaluated MOSAIC-compliance for the HTTP video streaming services shown in Table II.

The YouTube site provides video via progressive download in either Flash or HTML5 format. For the Flash format, the Chrome browser invokes the Flash Video Player plug-in while it natively supports HTML5 format. For both formats, YouTube did support partial GET.

Note that this trial scenario confirms the need for application transparency since MOSAIC could support both applications, i.e., the video player plug-in as well as the browser's native video support.

TED, a website for oral presentations with educational content, also supports progressive download via Flash and HTML5 format. The Flash-based service, however, is based on the Real-Time Messaging Protocol (RTMP) [21], which is a proprietary state*ful* protocol by Adobe. Stateless mobility is therefore not supported. The HTML5-based service is restricted to Apple's iOS devices. In order to receive the HTML5-based video for a Linux client, we had to change the Chrome browser's User-Agent setting to "iPad". We could confirm MOSAIC-compliance for the HTML5-based service.

We evaluated two different variants of Dynamic Adaptive Streaming over HTTP (DASH) namely Adobe HTTP Dynamic Streaming [22] and Microsoft Smooth Streaming [23]. These services require the respective Adobe Flash and Microsoft Silverlight plug-ins. We could confirm that both services supported partial GET. Further, MOSAIC properly worked for both services, which verifies MOSAIC's applicability to DASH.

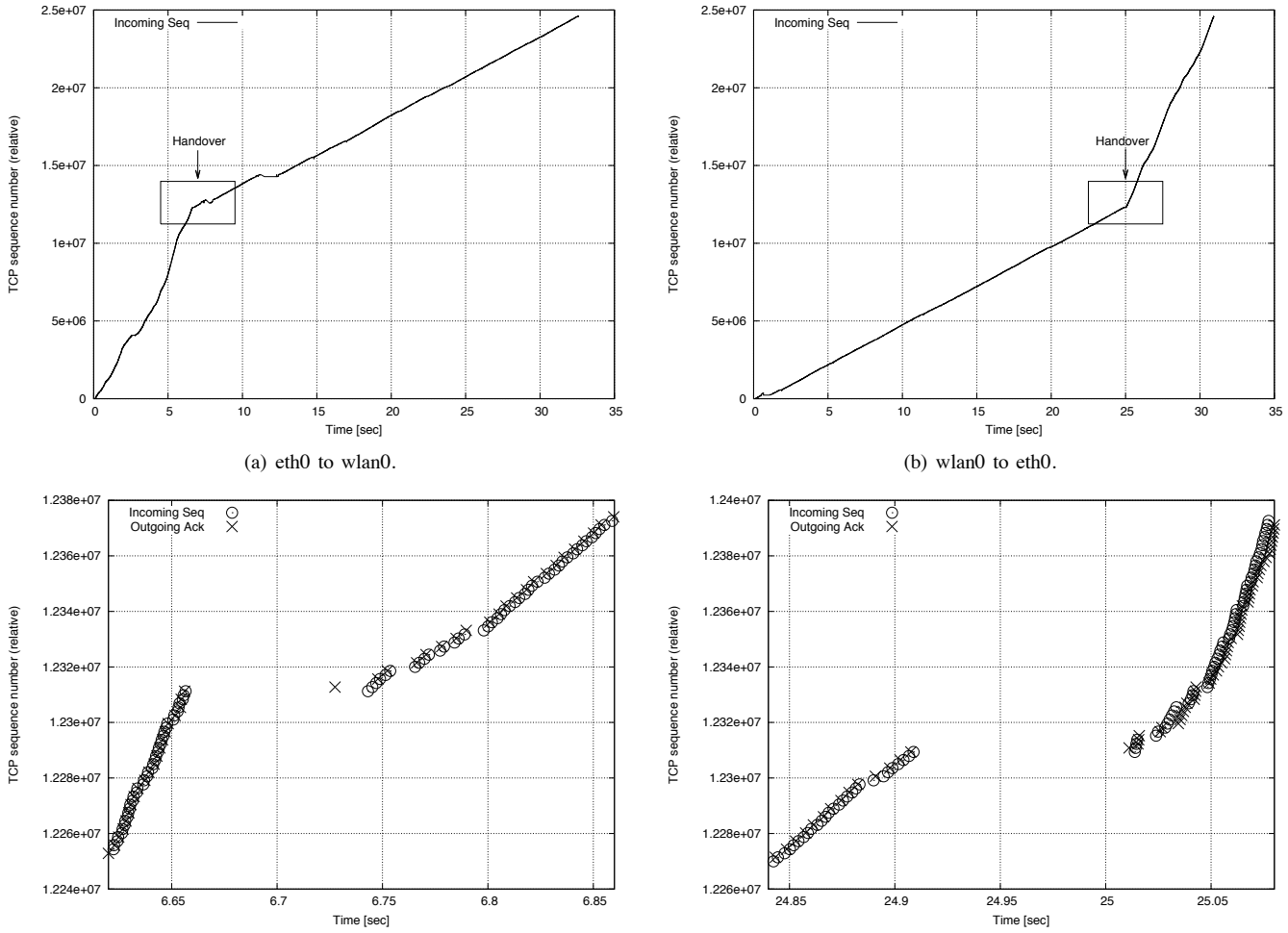
Unfortunately, we could not evaluate Apple HTTP Live Streaming [24] since there is no QuickTime plug-in for Linux. We also could not evaluate Netflix [25], the most prominent use of Microsoft Smooth Streaming technology, since Netflix uses the Digital Rights Management (DRM) feature of Silverlight, which is not supported by Moonlight.

### B. Handover Evaluation

We performed a detailed analysis of MOSAIC's behavior at mobility events.

Figure 4 shows the measurements of packet sequence number plotted against time for a TED video playback. Figure 4(a) depicts a handover event from Ethernet to Wi-Fi and Figure 4(b) a handover event from Wi-Fi to Ethernet. The handovers occurred at 6.6 second and 24.9 second, respectively, creating a slope change in the measurement data.

The different slopes in the graphs of Ethernet-based and Wi-Fi-based transport reflect the different throughputs of the respective access technologies. The small plateau regions between 11.0 and 12.4 second in Figure 4(a) indicate packet loss and retransmission events.




(a) eth0 to wlan0.

(b) wlan0 to eth0.

(c) eth0 to wlan0 zoomed.

(d) wlan0 to eth0 zoomed.

Fig. 4. Time vs. TCP sequence number graph.

Figures 4(c) and 4(d) show a zoom-in of a small time interval around the respective handover events. These plots clearly indicate that each handover event is associated with a time delay. We provide a detailed analysis of the handover delay in Section V-D.

The sequence numbers on the ordinates of these figures pertain to the host's TCP control block. The ordinate has been normalized to the initial sequence number used at session start. Obviously, the sequence number stream remains contiguous through the handover events. This shows that MOSAIC's L3 implementation works properly since it makes mobility events transparent to the mobile host's transport layer.

TABLE III
DESTINATION IP ADDRESSES AFTER HANDOVER.

| Trial number | Interface | Destination IP address |
|---|---|---|
| 1 | eth0 | 67.148.147.43 |
| | wlan0 | 67.148.147.43 |
| 2 | eth0 | 67.148.147.43 |
| | wlan0 | 67.148.147.43 |
| 3 | eth0 | 67.148.147.43 |
| | wlan0 | 67.148.147.129 |
| 4 | eth0 | 67.148.147.120 |
| | wlan0 | 67.148.147.129 |
| 5 | eth0 | 67.148.147.120 |
| | wlan0 | 67.148.147.120 |

### C. Change of Content Server

We verified MOSAIC's ability to change the content server during a mobility event. Table III shows the IP addresses of the video streaming servers used for a TED video playback before and after five interface handover trials. While both IP addresses of each trial were obtained via DNS, the first IP address was selected by the video client on the Ethernet link and the second by MOSAIC on the Wi-Fi link. When the DNS response returned multiple IP addresses, MOSAIC always selected the first candidate. The selection criteria applied by the video client were not known.

In trial 3 and 4, the remote content server was indeed changed during the handover event. In trial 1, 2, and 5, however, the content server after handover was the same as before. This indicates that the subnets are topologically very close to each other (they may in fact belong to the same

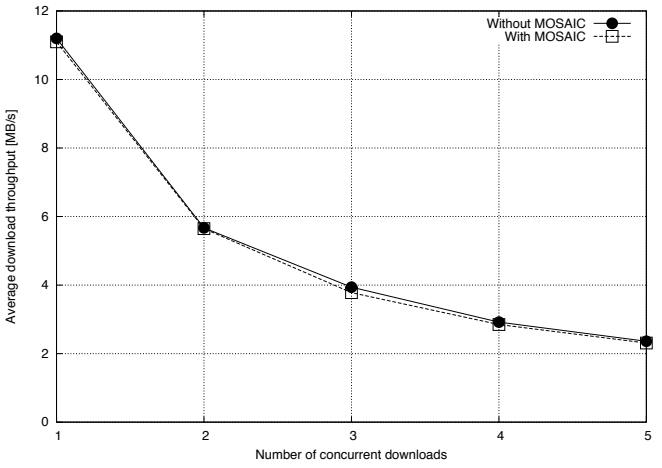



Fig. 5. Average download throughput with and without MOSAIC.

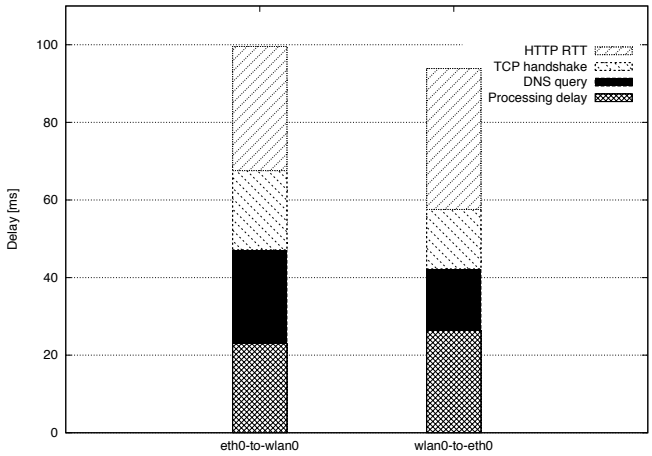



Fig. 6. Handover delay microbenchmark.

administrative domain). It further shows that the video client uses the same IP selection criteria as MOSAIC. The content server changes observed in trial 3 and 4 are most likely due to a different IP-address prioritization in the DNS response.

The outcome of these trials was positive since the video sessions properly continued across the mobility events in all five cases.

### *D. Performance Impact*

We evaluated MOSAIC's impact on throughput as well as its handover delay.

Figure 5 compares the average throughput of TED video downloads with and without MOSAIC as we increase the number of concurrent downloads. At three concurrent downloads, for example, the throughput with MOSAIC is 4% lower–3.78 MB/s versus 3.94 MB/s. The graph indicates that MOSAIC's impact on throughput is negligible despite the fact that MOSAIC inspects all HTTP packets.

The handover delay is defined as the time between the handover request by the connection manager and the reception of the first content packet on the new interface. This delay applies to multi-homed hosts, where both interfaces are simultaneously available when handover occurs. It does not include delay associated with the host's establishment of L2 connectivity or DHCP handshake to obtain a new IP address.

The total handover delay was analyzed with respect to the various contributions such as processing delay, DNS query, TCP handshake, and consecutive HTTP handshake. The processing delay is mostly caused by the update of the host's routing table.

Figure 6 shows average results for Ethernet-to-Wi-Fi and Wi-Fi-to-Ethernet handovers. The total delay stays slightly below 100 ms. This value can be reduced to 60-70 ms if DNS query and TCP handshake on the new interface are conducted *before* the old link is discontinued. Since this would require replacing the explicit changes to the routing table with source routing, it may also reduce the processing delay.

In any case, a handover delay less than 100 ms may be acceptable for most applications. Especially for video streaming, where we expect MOSAIC to have the largest benefit, such a delay will be covered by the playback buffer provided by most video players [26].

## VI. Discussion

Our trials demonstrate that MOSAIC provides proper mobility support for actual websites as long as these services implement the partial GET feature. This applies to the principal concept of stateless mobility for HTTP-based applications as well as the application-transparent implementation we conducted. We further verified MOSAIC's applicability to HTTP adaptive streaming services, whose traffic volume is expected to substantially grow over the next years [27].

The trials further indicate that stateless mobility is supported by all those services, where mobility support may be of value since content retrieval consumes an extended amount of time. This applies to services such as file transfer and video streaming. This finding supports the market readiness for the deployment of stateless mobility. Since MOSAIC only requires changes to the mobile host, rollout could occur rather fast and at little cost. In this context, the demonstration of a packet-filter-based implementation may come in handy since it provides easy portability to other platforms.

While these findings cast a positive light on stateless mobility, there are inherent limitations to its principal concept. Firstly, stateless mobility only applies to stateless applications. This limitation may not seem too severe since the fraction of traffic due to stateless applications is on the rise. It may be worth, however, to also investigate traffic transfers from the mobile to the cloud, which may gain in importance and represent the equivalent of a mobile server communicating with a network-side client.

Secondly, stateless mobility is not applicable to time-sensitive content since content retrieval started at one time cannot be continued with a new request at a later time. This, however, is not a principal limitation since time-sensitive

content can be made time-invariant via HTTP redirections to time-sensitive URLs. Since the inherent problem of time-sensitive content also applies to information-centric networks, we expect this topic to find sufficient resonance within the research community.

Thirdly, MOSAIC cannot serve HTTPS-secured traffic in an application-transparent manner. This limitation is due to the encapsulation of HTTP requests, which makes it impossible for MOSAIC to retrieve the URL contained in an HTTP request or to add an additional `Range` header. To overcome this limitation, MOSAIC would have to be integrated into the host's TLS library.

There are a few issues that remain to be verified through trials. One of them is MOSAIC's proper operation in presence of HTTP redirection discussed above. Another is MOSAIC's operation in the presence of HTTP proxies. It would further be of value to conduct mobility trials in real-world scenarios, e.g., handovers between licensed and unlicensed spectrum pertaining to different operators.

## VII. Conclusion

We introduced the concept of stateless mobility and presented a prototype implementation. Our implementation is a generic solution for all HTTP-based applications and can be easily ported to mobile platforms. The trials we conducted with this implementation confirm the overall validity of our proposal across a number of scenarios, and they quantify or at least bound processing and performance overhead.

## References


[1] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian, "Internet Inter-Domain Traffic," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 75–86, 2010.

[2] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[3] G. Peng, "CDN: Content Distribution Network," State University of New York at Stony Brook, Tech. Rep. TR-125, 2008.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, June 1999.

[5] Y. Nkansah-Gyekye and J. Agbinya, "Vertical Handoffs in Fourth Generation Wireless Networks," *Advances in Broadband Communication and Networks*, pp. 277–308, 2008.

[6] G. Naik, "LTE WLAN Interworking for Wi-Fi Hotspots," in *Proceedings of the Second International Conference on Communication Systems and Networks (COMSNETS)*, January 2010.

[7] C. Perkins, "Mobile IP," *IEEE Communications Magazine*, vol. 35, no. 5, pp. 84–99, 1997.

[8] J. Pan, Y. Hou, and B. Li, "An Overview of DNS-based Server Selections in Content Distribution Networks," *Computer Networks*, vol. 43, no. 6, pp. 695–711, 2003.

[9] L. Wang, V. Pai, and L. Peterson, "The Effectiveness of Request Redirection on CDN Robustness," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 345–360, 2002.

[10] V. Adhikari, S. Jain, Y. Chen, and Z. Zhang, "Reverse Engineering the YouTube Video Delivery Cloud," in *Proceedings of IEEE Hot Topics in Multimedia Delivery (HotMD)*, July 2011.

[11] Z. Zhu, R. Wakikawa, and L. Zhang, "A Survey of Mobility Support in the Internet," RFC 6301, July 2011.

[12] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," RFC 6182, March 2011.

[13] S. Gundavelli, K. Leung, V. Devarapalli, K. Chowdhury, and B. Patil, "Proxy Mobile IPv6," RFC 5213, Auguest 2008.

[14] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," RFC 4423, May 2006.

[15] A. Snoeren and H. Balakrishnan, "An End-to-End Approach to Host Mobility," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom)*, August 2000.

[16] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking Named Content," in *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, December 2009.

[17] T. Stockhammer, "Dynamic Adaptive Streaming over HTTP–Standards and Design Principles," in *Proceedings of the 2nd Annual ACM Conference on Multimedia Systems (MMSys)*, February 2011.

[18] C. Bernardos, "Proxy Mobile IPv6 Extensions to Support Flow Mobility," Internet Draft, draft-ietf-netext-pmipv6-flowmob-02, October 2011.

[19] "The Netfilter project," http://www.netfilter.org/.

[20] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, "Linux Netlink as an IP Services Protocol," RFC 3549, July 2003.

[21] "Real-Time Messaging Protocol (RTMP) specification," http://www.adobe.com/devnet/rtmp.html.

[22] "Adobe HTTP Dynamic Streaming," http://www.adobe.com/products/hds-dynamic-streaming.html.

[23] "Microsoft Smooth Streaming," http://www.iis.net/download/SmoothStreaming.

[24] "Apple HTTP Live Streaming," https://developer.apple.com/resources/http-streaming/.

[25] "Netflix," http://www.netflix.com/.

[26] K. Ma, R. Bartos, and S. Bhatia, "A Survey of Schemes for Internet-Based Video Delivery," *Journal of Network and Computer Applications*, vol. 34, no. 5, pp. 1572–1586, 2011.

[27] Cisco Systems, "Cisco Visual Networking Index: Forecast and Methodology, 2010-2015," June 2011.