A Multiple Care of Addresses Model

Bruno Sousa, Marco Silva
CISUC, University of Coimbra
Polo II, Pinhal de Marrocos
3030-290, Coimbra, Portugal
Email: bmsousa,mfsilva@dei.uc.pt

Kostas Pentikousis
Huawei Technologies
European Research Center
Carnotstrasse 4, 10587 Berlin, Germany,
Email: k.pentikousis@huawei.com

Marilia Curado
CISUC, University of Coimbra
Polo II, Pinhal de Marrocos
3030-290, Coimbra, Portugal
Email: marilia@dei.uc.pt

***Abstract*—Resilience, load balancing and ubiquitous support can be improved by multihoming configurations that explore the plurality of wireless technologies available nowadays. Nevertheless, efficient multihoming configurations require support from all layers, as for instance in, network protocols which must incorporate mechanisms to support multiple addresses. Multiple Care of Addresses Registration (MCoA) is a protocol that extends Mobile IPv6 to enable the registration of multiple addresses.**

This paper presents mCoA++, our publicly available simulation model for OMNeT++ which implements the Multiple Care of Addresses Registration protocol recently standardized by IETF. Filling a gap in the MCoA specification, mCoA++ incorporates cross-layer mechanisms that tailor address selection according to application requirements. We evaluate our mCoA++ implementation and compare simulation code performance with xMIPv6. We find that mCoA++ adds multiple care of address support in OMNeT++ without introducing any significant overhead.

Keywords – MIPv6, Multiple CoA, Multihoming, OMNeT++, mCoA++, mobility modelling, and performance

I. Introduction

Multihoming support is emerging as a strong feature for modern devices which typically move in areas with overlapping coverage. Multiple Care of Address Registration (MCoA) [1] is a recent recommendation which specifies the registration of multiple addresses enabling multihoming in MIPv6 nodes.

Different approaches may be followed to explore multiple addresses in mobile networks. HIPSim++ [2] is an implementation of the Host Identity Protocol (HIP) in OMNeT++, that supports multiple addresses. Nevertheless, HIP is not compatible with MIPv6, thus MIPv6-aware nodes have no mechanism to explore multiple addresses. Another implementation explores multipath at the transport layer [3], by extending the Stream Control Transport Protocol (SCTP) to support concurrent multipath transfers. Once again, this solution does not allow MIPv6 nodes to support multiple addresses without the assistance of another protocol such as SCTP.

The Capacity-aware preferred Multiple Care of Address (CAPMCoA) [4] allows a mobile node to choose a CoA from the several addresses, based on the best throughput of a specific link-address pair. CAPMCoA does not meet requirements of today's applications, since it only considers a throughput metric. That is, applications interested in paths with low delay have no benefit with CAPMCoA. In addition the implementation is not publicly available and refers to an old version of the MCoA specification [5]. Whilst the WIDE project [6] adds Multiple Care of Address support to NEMO [7] implementation, it is based on an old version of MCoA.

Resorting to the related work, we found that there is no public and up to date implementation of MCoA [1] in common networks simulators (e.g., ns2, OMNeT++). Thus, we embarked on developing mCoA++ for OMNeT++ based on xMIPv6 [8], as it is an accurate implementation of MIPv6. Moreover, xMIPv6 and mCoA++ are implemented in OMNeT++ [9], a discrete event simulator that is gaining community attention, due to its possibility of including frameworks for different purposes (e.g., Ad-Hoc, wireless sensor networks, etc).

mCoA++ is the first implementation of MCoA in OMNeT++. The contributions of this paper and mCoA++ are four-fold: First, mCoA++ is an implementation that is freely available [10] to the community for further development/enhancement and use. Second, mCoA++ implements the MCoA protocol based on the lastest specification [1] and includes support for two major types of address use, *ALL*- all the addresses are used simultaneously and *SINGLE*- current address is chosen randomly or from the first advertisement received. Third, mCoA++ comprises mechanisms that enable address selection synchronisation between application and network layers, as this improves multihoming support [11]. Fourth, this paper presents how mCoA++ can be used and presents proof-of-concept results for the implementation and how mCoA (i.e. OMNeT++ implementation: mCoA++) compares with MIPv6 (i.e. OMNeT++ implementation: xMIPv6).

Potential users of mCoA++ include researchers working in multihoming, who aim to perform comparisons between MCoA and other approaches. mCoA++ can be extended to include other proposals, as it is freely available, and can be used to test applications in multihoming environments or to mitigate flow issues in Mobile IPv6 networks.

This paper is organized as follows: Section II provides an overview of the MCoA protocol. Section III details mCoA++ model and Section IV describes our evaluation methodology. Section V details the accuracy results of mCoA++ implementation and Section VI presents application performance results. Section VII concludes this paper.

II. Multiple Care of Address

Mobile IPv6 performs the binding of a single care of address (CoA), which is a limitation for Mobile Nodes (MNs) with multiple interfaces/addresses. The Multiple Care of Address Registration (MCoA) protocol [1] addresses this limitation by extending Mobile IPv6 to support the registration of multiple Care of Addresses (CoAs). MCoA introduces a new Binding Identification (BID) number to identify bindings, thus allowing multiple CoAs to be bound to the home address.

MCoA also introduces enhancements in the Binding Update (BU) messages to include the Binding Identifier Mobility Option that contains the BID(s) to register. On the reception event of a BU message, the Home Agent (HA) and/or Correspondent Node (CN) create or update the respective bindings in the Binding Cache (BC). To support multiple bindings for a home address, the Binding Cache lookup is performed by the home address and the BID pair, as opposed to MIPv6 that relies only on the home address. If the HA or the CN do not support registration of multiple addresses, they acknowledge the MN in the Binding Acknowledge (BA), so that in the next message exchange MN resorts to standard MIPv6.

The Binding Identifier Mobility Option is included in the Binding Acknowlegment (BA), Home Address Test (HoT) and Care of Address Test (CoT) messages. This option includes several fields, such as the length that depends on the IP version (e.g. IPv4 or IPv6), the Binding ID and the status field, which reports the registration state of the respective CoA.

When the MN wants to register, it generates a Binding ID (value between 1 and 65535) per address, and sends a BU message to HA and CNs, keeping the BID in the Binding Update List (BUL). When the MN has several addresses to register, it can use the bulk registration mode that includes several Binding Identifier mobility options in a single BU message. The bulk registration, where a single BU message conveys multiple BID options, is only supported with the HA. Thus, the registration with the CN must be performed per address, to avoid issues with the return routability procedure.

We note that MCoA does not specify how the multiple registered addresses can be used. For instance, if the addresses can be used simultaneously or if a Care of Address is chosen based on link characteristics. This open specification can be tailored to the specific application requirements. For instance, real-time applications are interested in a link/CoA with smaller end-to-end delay, while data applications in a link/CoA with higher (nominal) capacity.

III. mCoA++: Multiple Care of Address Model

We start the overview of our implementation of Multiple Care of Address Registration in OMNeT++[9], which we dubbed as mCoA++, with design considerations.

Our mCoA++ implementation aims at two major goals: mCoA++ ought to be RFC 5648 [1] compliant and should not break the compatibility with MIPv6. To satisfy the latter goal, we started by deriving mCoA++ from xMIPv6 [8] implementation in OMNeT++. xMIPv6 was chosen because it implements all the features of MIPv6 for mobility management (e.g. tunnel creation/modification/deletion) and it is a flexible framework for easy extensions. As mentioned above, the way the multiple addresses can be used is not specified in RFC 5648. Taking this into account, we implemented two major types of use, namely, *ALL* and *SINGLE*. In both cases all interface addresses are registered and a separate tunnel is created for each address. However, with *ALL*, applications use the several addresses simultaneously by replicating packets for each tunnel, while with *SINGLE*, only one address is used.

Mobile IPv6 informs upper layers (e.g. applications) when tunnels are created/deleted to assure that the addresses chosen by applications are valid (reachable through a certain path). The cross-layer mechanisms, herein implemented rely on the notification schemes of the *INET* framework, which implements protocols such as SCTP and IPv6.

In our implementation the application chooses the type of use. For instance, data applications may be interested on addresses associated with paths with higher bandwidth, while VoIP applications are interested on paths with reduced end-to-end delay. This approach is inline with recent proposals for cross-layer design, including architectures based on IEEE 802.21 (http://www.ieee802.org/21/), for example. Thus, we expect that mCoA++ can be used as valuable building block in simulation studies of cross-layer architectures and evaluations of mobility management solutions, among others.

A. Classes and Nodes

Table I summarizes new classes introduced and modified ones in mCoA++. The class `MCoA` is added in the network layer of the MN, HA, and CN nodes. This class/module has no connections to other modules since no messages are exchanged between them and to keep the compatibility with xMIPv6 model. The MCoA class works as a configuration class, and the respective configuration directives are implemented in the `xMIPv6` class, for instance in `SendPeriodicBU()`. Moreover, the class `MCoA` configures several MCoA parameters in MIPv6 aware nodes. The *m_prohibited* indicates if a node supports the registration according to MCoA. Standard MIPv6 corresponds to $m\_prohibited = true$. The *m_bulk_reg_prohibited* flag indicates if a node supports bulk registration. The *mc_sim_home_and_foreign_prohibited* flag allows the simultaneous use of home and foreign interfaces. The *TypeUseMCoA* is a string field to define the type of use to employ for the registered addresses. The possible values defined in the `MCoADefs.h` file, include *ALL*, *SINGLEFIRST* and *SINGLERANDOM*. Moreover, the *deregisterALL* is an integer field to indicate how the deregistration should be performed (e.g., 1 to deregister one-by-one).

The `xMIPv6` class is the core class of the Mobile IPv6 implementation, as such major features for MCoA support were coded in this class. Several data structures were modified to accommodate information about BIDs. For instance, `KeyTimer` and `InterfaceCoAList` were modified to include the BID field, as well as the respective search methods. In addition, new data structures were introduced to hold the

Class	Purpose
KeyMCoABind	Key in the Binding Update List (BUL) and Binding Cache (BC).
KeyMCoADAD	For operations with Duplicate Address Detection (DAD).
XMIPv6SM	State Machine for MIPv6 operation.
IPv6TunAdr	Information for created/deleted tunnels.
IPv6PrefAdr	Information about preferred address.

Class	Modification detail
BindingCache	Introduce new key, KeyMCoABind.
BindingUpdateList	Introduce new key.
IPv6Tunneling	Include notification of created/deleted tunnels.
IPv6	Allow activation and de-activation of IPv6 forwarding to simulate errors at the network layer.

TABLE I: Classes in mCoA++

tuples of CoA and BID pairs - CoABIDList and to hold the node supported MCoA capabilities - NodesMCoACap. New methods were introduced to accomplish different tasks. For instance the get_and_calcBID() is the method responsible for the BID assignment, implementing rules to avoid the Home address of MN to be included in the CoABIDList. The method get_adr_from_bid() retrieves an address from a BID number. The getMIPv6CoA() method is responsible for the active address selection, which takes into consideration if the node has initiated returning home operations, and the type of use to employ to the multiple care-of addresses.

When MIPv6 is initiated, timers to enable the creation of bindings are created (e.g. *KEY_BUL*). These timers are created for the HA and for each CN (identified in the *CNListBID*). When sending Binding Updates, sendPeriodicBU() method, the MCoA support is checked, which restricts the fulfillment of MobilityBIDOptions. The destination of binding messages is considered, therefore the set_mobilityoptions_for_ha() method is employed if registering with the HA while set_mobilityoptions_for_cn() method is used when registering with the CN. These methods set the MobilityBIDOptions according to the BIDs defined in the CoABIDList. On the BU message reception, HA and CN process this message type, checking each option in the MobilityBIDOptions. Timers to expired entries, refresh request timers are created, as well as the respective tunnels. MN is informed about the registration operation in the HA and CN through BA messages, which also convey the Mobility Options. MN proceeds according to the status of the BA message. On a successful binding operation, MN updates the BUL and initiates the return routability procedure for each BID, if the BA message comes from the HA. Tunnels are created according to the source of the BA message.

The returning home operation involves several operations (e.g., remove addresses, deregister BID) that are triggered in the returningHomeMCoA() method. When deregistering, the tunnels are removed and the timers are created with $bindinglifetime = 0$. The state machine XMIPv6SM is also updated. In order to allow the MN to roam again, a notification message is employed to guarantee that all the auxiliary structures and respective states are initialized.

B. Headers and Mobility Options

mCoA++ includes support for the Binding Identifier Mobility Option, MobilityBIDOptions. This option includes a status field for the BID, a flag to indicate if it is a home binding *Hflag*, and the respective BID key.

Messages that can include information about BIDs were modified to include support for a vector of Binding Identifier Mobility Options, such as BU, BA, *Home Test Init (HoTI)*, *Home Test (HoT)*, *Care of Test Init (CoTI)*, *Care of Test (CoT)*, *Binding Refresh Request (BRR)* messages.

C. Notifications

The notification NF_MIPv6_MN_RETURNED_HOME is subscribed by the MIPv6 class, which sends a message to perform cleaning operations after returning home.

In order to allow applications to be acknowledged about routing optimization that exists with correspondent nodes, the NF_IPv6_TUNNEL_ADDED and NF_IPv6_TUNNEL_DELETED notifications were added, so that an application could know when all the routing processes, at the network layer, are ready to forward traffic via optimized routes. Thus, a node is notified when a tunnel is created and deleted, respectively.

Also, in a cross-layer fashion, applications can set the preferred address they intend to use. The NF_MCOA_APP_PREFERED_ADDRESS notification is used to inform the network layer about the preferred address selected by applications and the respective priority.

D. MCoA Application Support

In order to receive notifications about the creation of tunnels and to utilize the information received on such notifications, applications have to be extended. The class MCoAUDPBase is a base class for UDP applications. This class contains information about sockets, namely source, destination addresses and sockets id, adrsAvailable. The class MCoAUDPBase implements methods for diverse socket bindings and unbindings, and methods for sending packets according to the type of use configured. In the method sendToUDPMCOA(), the application sends packets according to the $typeUse$:

- ALL, packet is duplicated for n addresses.
- $SINGLERANDOM$, preferred address is chosen randomly from the adrsAvailable vector.
- $SINGLEFIRST$, packets are sent to the first address.

The MCoAUDPBase class works as a base class that can be used by other applications. Initially it performs binding to the default port and addresses configured, but when receiving notifications about the creation and deletion of tunnels, it performs the socket binding using the information received with the notification messages (e.g., source, destination addresses). On the deletion operation, the socket (identified by an integer) is marked as deleted to avoid using this socket on future events.

The MCoAUDPBase class includes various parameters that affect the MCoA operation. The *localPort* on which socket

List 1: MCoA application example code

```
1  int sockID=bindToPort(localPort, ipSrc_Address);
2  // msg = new cPacket();
3  if (useMode == MCOA_TUN_ALL_ADR_SINGLE_RR){
4    int idx = (int)intrand(lenAdrs);
5    IPvXAddress adrtoSend = adrsAvailable[idx].mSrc;
6    sendToUDP(msg, adrtoSend, srcPort,
7            destAddr, destPort, appendCtrlInfo);
8  }
9  // Unbind operation
10 unBindPort(localPort, ipSrc_Address, sockID);
```

List 2: Configuration Example

```
<interface name=''eth1'' AdvSendAdvertisements=''on''>
 <AdvPrefixList>
   <AdvPrefix AdvOnLinkFlag=''on'' AdvValidLifetime=''4''
     AdvPreferredLifetime=''4'' AdvAutonomousFlag=''on''
   advRtrAddr=''on'' rtrAddr=''2001:db8::0299:2B2''>
     2001:db8::0299:00/112</AdvPrefix>
   <AdvPrefix AdvOnLinkFlag=''on'' AdvValidLifetime=''4''
   AdvPreferredLifetime=''4'' AdvAutonomousFlag=''on''
   advRtrAddr=''on'' rtrAddr=''2001:db8::0199:2B2''>
     2001:db8::0199:00/112</AdvPrefix>
 </AdvPrefixList>
 <inetAddr tentative=''off''>
  2001:db8::0299:2B2 </inetAddr>
 <inetAddr tentative=''off''>
  2001:db8::0199:2B2 </inetAddr>
</interface>
</local>
```

bindings should be done, the possible *destAddresses*, the *useMode* parameter that dictates the type of use (values according to `MCoADefs.h`) and the *isDestiny* flag indicates if the node is acting as receiver `true`, or sender `false`.

Applications needing *MCoA* facilities need to extend `MCoAUDPBase` class and implement their own sending mechanisms, (sending rate, packet size). The `MCoAVideoStreamCli`, `MCoAVideoStreamServer` implement a video MCoA-capable application, and `MCoAUDPApp` implements a VoIP MCoA-capable application with the possibility of requesting echo of messages.

List 1 provides some excerpts of code that exemplify the creation and use of MCoA in UDP applications. Line 1 depicts the binding to a local port. The `bindToPort` returns a socket ID for future operations (e.g. unbinding operation). Lines 4-5 exemplify random address selection. Line 6-7 call the send method that appends control information to the message (source and destination addresses). The last line illustrates the deletion of socket, by performing the unbinding operation. Further details on the usage can be found on the applications `MCoAVideoStreamCli`, `MCoAVideoStreamServer` and `MCoAUDPApp` that are included in the mCoA++ code [10]. The first implementation of mCoA++ only included UDP applications, future releases will accommodate support for TCP applications, as well.

E. Configuration

The current `FlatNetworkConfigurator6` module configures the whole simulation as a big subnet. In addition, it assigns a simple prefix per router. To enable the advertisements of several prefixes by a router, and to have distinct networks, the `MCoANetConf6` module was introduced. This module implements `addOwnAdvPrefixRoutes()` and `addStaticRoutes()` methods to add the advertisement prefixes and static routes, for routers and hosts, respectively. All prefixes can be configured in a XML file, as illustrated in List 2. This implementation provides flexibility to configure more realistic network scenarios.

IV. Evaluation Methodology

The evaluation has two purposes: First, validate the mCoA++ model through a comparison with xMIPv6. Second, demonstrate the wide-applicability of mCoA++ in multihoming context (e.g., several addresses).

The simulation scenario includes the correspondent node network, the home network and two foreign networks. Multiple prefixes are advertised on the foreign networks (*Subnet #1*, and *#2*). Moreover, router *R2* is connected to both networks, and router *R1* advertises multiple prefixes on *Subnet #1*. The Wi-Fi technology (IEEE 802.11b) is employed due to its wide employment. The ethernet connections are configured with a transmission delay of *10ms*, while the links simulating an Internet connection, between routers *Ra-Rb* and routers *Ra1-Rb* have a propagation delay of *30ms*. Internet links have associated higher propagation delays in comparison to Ethernet links. The scenario also encompasses Video and VoIP applications. Video transmits at a constant rate of $50ms$ with a packet size of $500B$, to simulate video streaming, while VoIP applications are set with packet interval of $10ms$ (G.723.1).

MN velocity was considered with 3km/h and 30km/h speeds, in order to emulate pedestrian and vehicular speeds. In addition, the rectilinear, *Rect*, and random way point, *Rwp*, mobility models were considered. Network failures were introduced, to assess the advantages of using multiple addresses, namely resilience. The network failures include disruption of IPv6 forwarding facilities with a duration of $150ms$ and are generated each 20s between *R1* and *R2* routers.

Different ways of using the multiple available addresses were considered. *MCoA_ALL* uses all the addresses simultaneously, *MCoA_SINGLE_FIRST* chooses the first CoA, *MCoA_SINGLE_RANDOM* chooses a CoA randomly from the several that are available, while *MIPv6* corresponds to the standard Mobile IPv6. In addition the bulk registration mode is enabled for the registration procedures with MCoA cases.

We compare the results (from 100 runs) achieved with MCoA and MIPv6, since MIPv6 accuracy has already been demonstrated on the xMIPv6 implementation [8], through different tests. As in the xMIPv6 [8] and Mobility Management Simulation Engine for IPv6 (MMSEv6) [12] we used the time of handover T_{HO} and the signalling cost metrics. The time of handover T_{HO}, includes delay between movement detection and node registration at the CN t_{CR}. We measured T_{HO} according to the following: $T_{HO} = t_{CR} - t_{ASSOC}$, in order

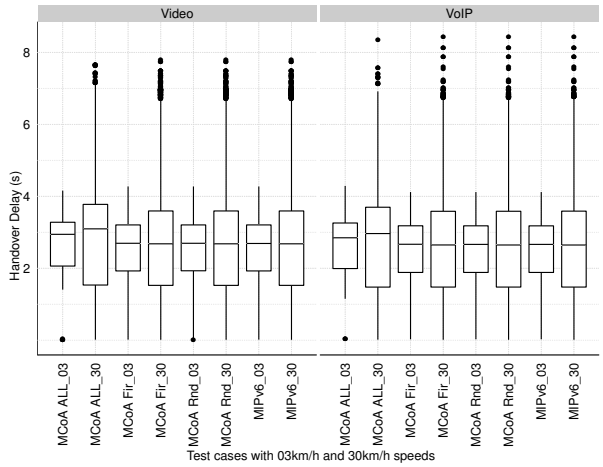

Fig. 1: Handover Time for Video and VoIP applications on MCoA ALL, MCoA ONE First (*MCoA Fir*), MCoA ONE Random (*MCoA Rnd*) and MIPv6 test cases.

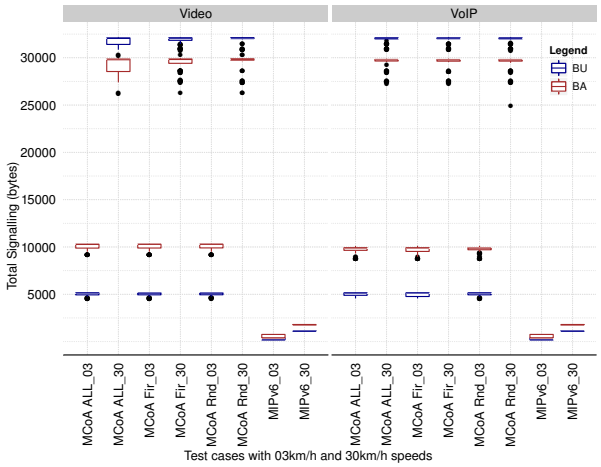

Fig. 2: Total signalling cost for Video and VoIP applications on MCoA ALL, MCoA ONE First (*MCoA Fir*), MCoA ONE Random (*MCoA Rnd*) and MIPv6 test cases.

to be agnostic of layer 2 handover delay. t_{ASSOC} corresponds to the instant on which the MN associates with an Access Point, t_{CR} corresponds to the time where tunnels are created or destroyed due to the reception of a successful registration (receive BA).

The signalling cost is based on the total signalling cost, which corresponds to the sum of the message size of the most common signalling MIPv6 messages, namely BU, BA, CoTI and HoTI. The message size, in bytes, includes header(s) size and respective payload size.

V. MCoA++ Performance

All results include a confidence interval of 95%. We compare handover latency results from mCoA++ implementation with the xMIPv6 implementation [8]. To assess mCoA++ performance, router advertisements are configured with min = 0.03s and max = 0.07s, a similar set of xMIPv6 evaluation. Fig. 1 shows handover delay for Video and VoIP applications in different speeds and configuration tests. Single test cases of mCoA++ have similar delay and are equal to the delay in the MIPv6 test cases. This corresponds to no performance degradation in mCoA++, in comparison to xMIPv6. regarding the handover delay. The mean delay in the MCoA ALL test cases is higher then in MCoA single and MIPv6 tests. The registration of multiple addresses and their simultaneous use require the establishment of tunnels for each address. While in MCoA single and MIPv6 cases only one tunnel is established in the MCoA ALL test cases multiple tunnels are created, one per each registered address, consequently handover delay is higher. The 30km/h cases have more handovers, around ~ 12 in opposition to ~ 2 for 3km/h speed cases. In all the test cases with 30km/h the handover delay has more variations (e.g., increased area in boxplots).

Despite the gain in Resilience (see Sec. VI), MCoA has drawbacks, namely in the signalling cost. The modified signalling messages in the MCoA specification include Mobility Options, which carry information for each address to register. The single binding nature of MIPv6, only registers one care of address. MCoA, by enabling the registration of multiple

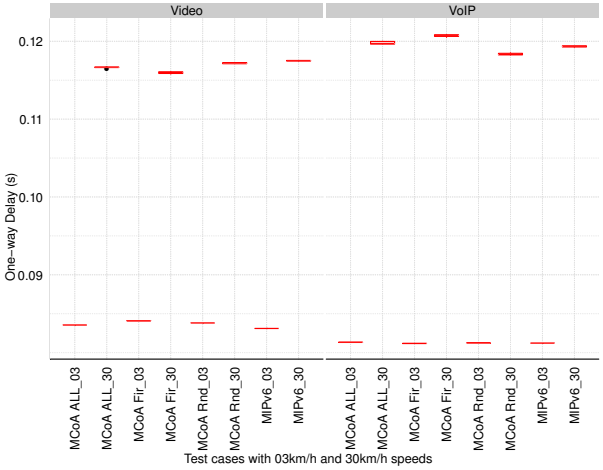

Fig. 3: One-way delay for Video and VoIP applications on MCoA ALL, MCoA ONE First (*MCoA Fir*), MCoA ONE Random (*MCoA Rnd*) and MIPv6 test cases

addresses, introduces more overhead, as per Fig 2. Signalling messages in mCoA++ convey multiple binding options, one per address. In addition, the number of handovers introduces more overhead as further messages are exchanged. Signalling cost between all the MCoA cases (ALL, ONE FIRST, ONE RANDOM) is similar since all the addresses are registered. As such, the several addresses are conveyed in the Mobility Options of signalling messages.

Both handover latency and signalling cost results depict the accuracy of mCoA++ implementation in comparison to xMIPv6. First, results are similar regarding handover latency. Second, mCoA++ introduces higher signalling costs.

VI. Applications Performance

Packet loss measurement relies on the lost message sequences. That is, each packet sent by the application is numbered and on arrival event, packet loss ratio is determined based on the sequences that were not received. This methodology was applied to VoIP and Video applications, since duplicated packets could lead to faulty results. Duplicated packets, for each tunnel in the *MCoA ALL* mode, do not affect

App.	Speed	MCoA ALL		MCoA ONE First		MCoA ONE RANDOM		MIPv6	
		Rect	Rwp	Rect	Rwp	Rect	Rwp	Rect	Rwp
VoIP	3km/h	1.43	15.90	1.43	15.90	1.44	13.72	11.32	17.39
VoIP	30km/h	35.11	55.06	34.80	58.69	35.08	56.52	52.66	63.60
Video	3km/h	1.40	16.49	1.41	17.13	1.43	16.49	17.53	20.90
Video	30km/h	35.33	56.65	35.35	57.30	35.33	56.74	56.49	64.67

TABLE II: Applications Packet Loss (%)

the payload (e.g. message sequence numbers), only headers (employed as control information) are modified, according to the respective tunnels.

To analyse the performance of applications, we can establish a relation between packet loss and resilience, by which higher packet loss ratios represent lower resilience levels. Within higher speeds (e.g. 30km/h) and with random way point mobility model, the level of resilience is minimized. Nodes can move to areas without coverage of Access Points, or have higher packet loss ratios due to increased handover delay at layer 2. MCoA-aware applications improve their resilience levels, even with higher speeds, as illustrated in Table. II for Video and VoIP applications. The registration of multiple addresses is the key of such improvement. We can argue, based on the achieved results, that MIPv6, due to its single-binding nature does not provide resilience support. With MIPv6, applications can have packet loss ratios around $\sim 17.5\%$ with low speeds and moving rectilinearly. With the single MCoA test cases (First and Random) the use of a fixed address (e.g., the first to be configured) does not provide any gain in the performance as it leads to higher packet loss when compared to the approach of choosing an address randomly.

One-way delay is determined by relying on message timestamps. Each message sent is also timestamped and on the reception event, one-way delay is calculated as being the difference between the received time and the message creation time. One-way delay, depends on the underlying technology. In addition, different channel propagation delays are associated to the fixed links, to simulate Ethernet and Internet connections, respectively. Fig. 3 depicts one-way delay results. With 3km/h all the tests have similar values, around $\sim 0.08s$, with increased speeds delay has some variations. In random and first mCoA++ cases, the address selected might be associated with a path with more failures. In addition, VoIP applications are more susceptible to higher speeds, due to their higher ratios.

MCoA can enhance application performance by increasing levels of Resilience or supporting optimized path selection mechanisms (e.g., select a path with lower end-to-end delay). Application performance results obtained with mCoA++ put in evidence two aspects: First, MCoA *per si* is not synonym of performance gain, for instance the cost of using all the addresses is higher. Finally, applications and network protocols must have synchronized path selection schemes to meet application requirements.

VII. Conclusion and Future work

This paper introduced mCoA++, our publicly available implementation of MCoA for OMNeT++ [10], and evaluated it comparatively with xMIPv6. The mCoA++ simulation model extends xMIPv6 significantly and enables the registration of multiple care of addresses. Our simulation results show that mCoA++ adds MCoA support in OMNeT++, without introducing any significant overhead when compared with the base xMIPv6 code for typical simulation scenarios.

We anticipate that the research community will find our contribution particularly useful. For example, using mCoA++ researchers can now model and evaluate networks with multiple address configurations in OMNeT++, something which is not possible with vanilla xMIPv6. Another example where mCoA++ can serve as a foundation is the evaluation of multimedia applications over multiaccess mobile networks, as well as the evaluation of flow mechanisms for mobile IPv6 networks. Our next steps in further developing mCoA++ include the implementation of Flow Bindings [13].

VIII. Acknowledgments

Bruno Sousa acknowledges the support of the PhD grant SFRH/BD/61256/2009 from Ministério da Ciência, Tecnologia e Ensino Superior, FCT, Portugal. The work has been supported by the TRONE project CMU-PT/RNQ/0015/2009.

References

[1] R. Wakikawa, V. Devarapalli, G. Tsirtsis, T. Ernst, and K. Nagami, "Multiple Care-of Addresses Registration," IETF RFC 5648, 2009.

[2] L. Bokor, S. Nováczki, L. T. Zeke, and G. Jeney, "Design and Evaluation of host identity protocol (HIP) simulation framework for INET/OMNeT++," in *MSWIM'09*. ACM, 2009.

[3] T. Dreibholz, M. Becke, J. Pulinthanath, and E. Rathgeb, "Implementation and Evaluation of Concurrent Multipath Transfer for SCTP in the INET Framework," in *SIMUTOOLS '10, ICST*, 2010, pp. 1–8.

[4] J.-Y. Pan, J.-L. Lin, and K.-F. Pan, "Multiple Care-of Addresses Registration and Capacity-Aware Preference on Multi-Rate Wireless Links," *AINA workshops 2008*, pp. 768–773, March 2008.

[5] R. Wakikawa, V. Devarapalli, G. Tsirtsis, T. Ernst, and K. Nagami, " Multiple Care-of Addresses Registration," IETF Internet Draft: draft-ietf-monami6-multiplecoa-03, 2008.

[6] R. Kuntz, "Deploying Reliable IPv6 Temporary Networks Thanks to NEMO Basic Support and Multiple Care-of Addresses Registration," *SAINT'07*, pp. 46–46, January 2007.

[7] V. Devarapalli and R. Wakikawa and A. Petrescu and P. Thubert, "Network Mobility (NEMO) Basic Support Protocol," IETF Request for Comments: 3963, January 2005.

[8] F. Z. Yousaf, C. Bauer, and C. Wietfeld, "An Accurate and Extensible Mobile IPv6 (xMIPV6) Simulation Model for OMNeT++," in *SIMUTools '08*. ICST , 2008, pp. 1–8.

[9] K. Wehrle, M. Günes, and J. Gross, *Modeling and Tools for Network Simulation*. Heidelberg: Springer, 2010.

[10] B. Sousa, "mCoA++: Multiple Care of Address Registration in OMNet++." [Online]. Available: http://mcoa.dei.uc.pt

[11] O. M. Raoof and H. Al-Raweshidy, "Game Theory and an Interface Selection Mechanism for Multi-homed Mobile Nodes," *ISCC*, pp. 616–623, Jul. 2008.

[12] F. Z. Yousaf, C. Müller, and C. Wietfeld, "A Comprehensive MIPv6 Based Mobility Management Simulation Engine for the Next Generation Network," in *SIMUTools '10*. ICST, 2010, pp. 1–8.

[13] G. Tsirtsis, H. Soliman, N. Montavont, G. Giaretta, and K. Kuladinithi, "Flow Bindings in Mobile IPv6 and Nemo Basic Support ," Internet Draft: draft-ietf-mext-flow-binding (work in progress), October 2010.