Modelling Mobility-Aware Applications for Internet-based Systems

Bruno Yuji Lino Kimura
Instituto de Matemática e Computação - IMC
Universidade Federal de Itajubá - UNIFEI
Itajubá - MG, Brasil
Email: kimura@unifei.edu.br

Edson dos Santos Moreira
Instituto de Ciências Matemáticas e de Computação - ICMC
Universidade São Paulo - USP
São Carlos - SP, Brasil
Email: edson@icmc.usp.br

***Abstract*—Future network architectures, such as Next Generation Network (NGN) and Vehicular Communication Network (VCN), will provide large scale mobile Internet access supported with multiple and small cells of wireless communication. Therefore, the access to the Internet services are prone to very frequent disconnections when the mobile node migrates across the cells. This paper deals with this concern and discusses issues involved on the modelling of mobility-aware applications. An abstract model is described by means of two Finite State Machines (FSM) designed for both mobile nodes, client and server. The FSMs are meant to networked applications to preserve their integrity in the event of frequent connection disruptions in Single Jumps (client side mobility) and Double Jumps (both client and server sides mobility). The FSMs for mobility-aware applications were evaluated using a handover delay mathematical model. With handover latencies from 230 ms in single jumps and from 450 ms in double jumps, simulation results show that the proposed strategies are capable of providing automatic Transmission Control Protocol (TCP) connection re-establishment with smooth impact when increasing the frame error rate and the data lost in handovers.**

Keywords-Mobility-Awareness; Application Layer Mobility; Internet-based systems; IP Mobility Management.

I. INTRODUCTION

The paradigm of Mobile Computing is already part of our modern lifestyle. However, new challenges are arising with future wireless networks. While NGNs consider numerous heterogeneous wireless networks with small cells to increase scalability, mobile nodes are becoming devices of higher mobility. Recently, Internet-based applications for safety and infotainment have been designed to be deployed in vehicles. These applications suffer frequent connection disruptions while the node migrates quickly among the road-side wireless Internet Protocol (IP) networks. These scenarios demand research efforts to enable efficient support for IP mobility in the TCP/IP protocol stack.

Given the common absence of network support for mobility, several solutions have been proposed to work at the various layers of the TCP/IP protocol stack. Mobile IP [1][2] is the general IP mobility solution in the literature. It is designed to work at the Network Layer to provide the special handling required in the addressing and forwarding configurations when a node is moved. Applications running on the mobile node do not need to deal with mobility. Such transparency is an important requirement for IP Mobility Management.

Considering upper layers (Transport and Application) and lower layers (Link and Network) to classify mobility protocols, the choice of the layer where the mobility handling should be implemented implies on where to put the inherent overhead:

i. *For the sake of transparency, mobility can be handled by the lower layers to reduce the impact on the upper layers*. This is achieved with additional infrastructures for managing location of mobile nodes and agents to forward incoming packets to the current location of the nodes [1][2]. However, specific infrastructure increases the cost of deployment and maintenance of the mobility solution. In addition, agents imply modifications in the core of IP networks.

ii. *Handling mobility at upper layers, then the mobile node takes control of its own mobility*. At the application level, host mobility implies the complete re-establishment of broken connections, peer authentication, and recovering data at the Socket buffers lost with disruptions [3][4]. The cost of deployment is reduced, since the handling is conducted end-to-end, therefore, there is no need for agents. On the other hand, it increases the overhead at the upper layer with the burden of the mobility.

In this sense, including mobility support in the TCP/IP stack is a dilemma. However, upper layer based solutions are less costly [5]: the support is provided without requiring a node to be attached to a Home Network, then there is no need for additional infrastructure; solutions are implemented as software components, so that their deployment and maintenance are easier; route optimization is inherent, since there is no entity to intermediate communicating along the end-to-end path; the mean-time-to-recovery (i.e., handover latency) takes no longer than a second [3], therefore, performance is also an advantage. Due to these reasons, several mobility protocols [3][4][6]–[10] are in favour of handling mobility at the upper layers.

There is a promising trend of designing communication

 ISBN: 978-1-61208-279-0

solutions to work at the Application Layer. With recent Web technologies, e.g., HTML5, in support for new paradigms, such as Cloud Computing and Internet of Things, the Web browsers became a generalized interface to access distributed resources on the Internet. In the meantime, efforts on Software-Defined Networking (SDN) have been made toward IP programmable networks by separating and implementing the traffic control as software service.

We advocate this trend and, in this paper, we discuss an abstract model for handling mobility at the Application Layer. Two Finite State Machines (FSM) are exploited to define distributed systems aware of communication events caused by the host mobility on the Internet. We aim to provide a general purpose mobility solution that can be implemented as a software service according to the computer architecture and language of choice of developers. We evaluated the performance of mobility-aware applications according to the handover delay model discussed in [11] and [12]. Simulation results show good performance when degrading parameters of quality of handovers, such as frame error rate and retransmission of data lost in mobility. Handover delays take no longer than 793 ms for the worst case.

This paper is organized as follows: in the next section , we describe the mobility-aware application model; Section III describes the handover delay mathematical model used to evaluate performance of the proposed mobility-aware applications; Section IV discuss results obtained from simulations using the handover delay model; and the last section brings our conclusions.

II. Mobility-Aware Application Model

We assume that the host mobility causes a failure F that breaks the TCP connections of a networked application. Failure F lasts a disconnection time, which is the handover latency $T_{handover}$, until the mobile node acquires Layer 2 and Layer 3 access at the visited network and the mobility protocol resumes ongoing communications. To provide seamless mobility the disconnection latency $T_{handover}$ should be as short as possible, which is a big challenge on Mobile Computing under research efforts in the last decade.

F is denoted by one of the following events:

$$F = \{\texttt{F:TO}, \texttt{F:RST}, \texttt{F:DU}, \texttt{F:BC}\}, \quad (1)$$

where they represent *connection timeout*, *connection reset*, *destination unreachable*, and *broken connection*, respectively. These events lead the system to unexpected states and, without the proper handling, they make the system crash.

We propose a repair plan at the application runtime to handle F events by means of two Finite State Machines, shown in Figure 1, regarding mobile clients connected to mobile servers. They identify unexpected states as erroneous states (grey circles). When reaching erroneous states, the system is able to lead to a safe state again (white circles) with transition operations based on the semantic of the Socket API, so that the communication is resumed consistently without losses. This system property provides mobility awareness to networked applications.

Next, we discuss fundamental aspects for handling mobility with the proposed model.

A. Session establishment

We assume that the mobile nodes are uniquely identified by an identifier decoupled from the IP address, as in [10]. IP addresses become locators that indicate the current logical location of the mobile nodes. When the (re)connection is established, we suggest a handshake for the nodes exchanging their local control information (`c.info` and `s.info`), as shown in the transitions c_3, c_4, c_5 and s_4, s_5, s_6. The control information represent a set of communication parameters that includes: host identifier, transmission checkpoints, and control flags that indicate the role (mobile or stationary node and client or server application) each communicating side plays in a session.

To generate a host identifier, we suggest the use of the SHA-1 algorithm to digest unique RSA Public Keys. The hash SHA-1 allows a low probability of bit collision as well as a huge name space of 2^{160} bits in length. This key material can be used for authenticating peers during handshake. This prevents spoofing and replay attacks during the (re)connections. The key material can be combined with a mutual challenge-question authentication [13], as we implemented in [3].

Thus, the server is aware of who the client is, and vice-versa, and also of where such (re)connection is coming from. Then, the server is able to abort (re)connections that come from untrusted clients, as shown in the transition from s_5 to s_{11}. In the meantime, the client denies communicating with fake servers by aborting connections in the transition from c_5 to c_{11}.

B. Saving the transmission status

After handover, the data enqueued in the send buffer may be lost. The lost data are those unconfirmed by the remote peer and those ready for sending. To provide reliability at the Application Layer, it needs to preserve a copy of the message to be sent, and checkpoints to successfully count the sent and received bytes.

When the peers are connected and ready for sending and receiving messages, the client is in state c_5 and the server is in state s_6. The client sends the message (`send(new.data)`) and reaches state c_6. Then, it saves the transmission context by keeping a local copy of the sent message (from c_6 to c_7) in an ancillary buffer, and it counts the amount of sent bytes (from c_7 to c_8).

After sending, a client can remain waiting for a server's response. When the socket receive buffer is ready for reading, the client consumes the enqueued data and reaches the state c_9 (from c_5 or from c_8). Then, it saves the transmission

 ISBN: 978-1-61208-279-0

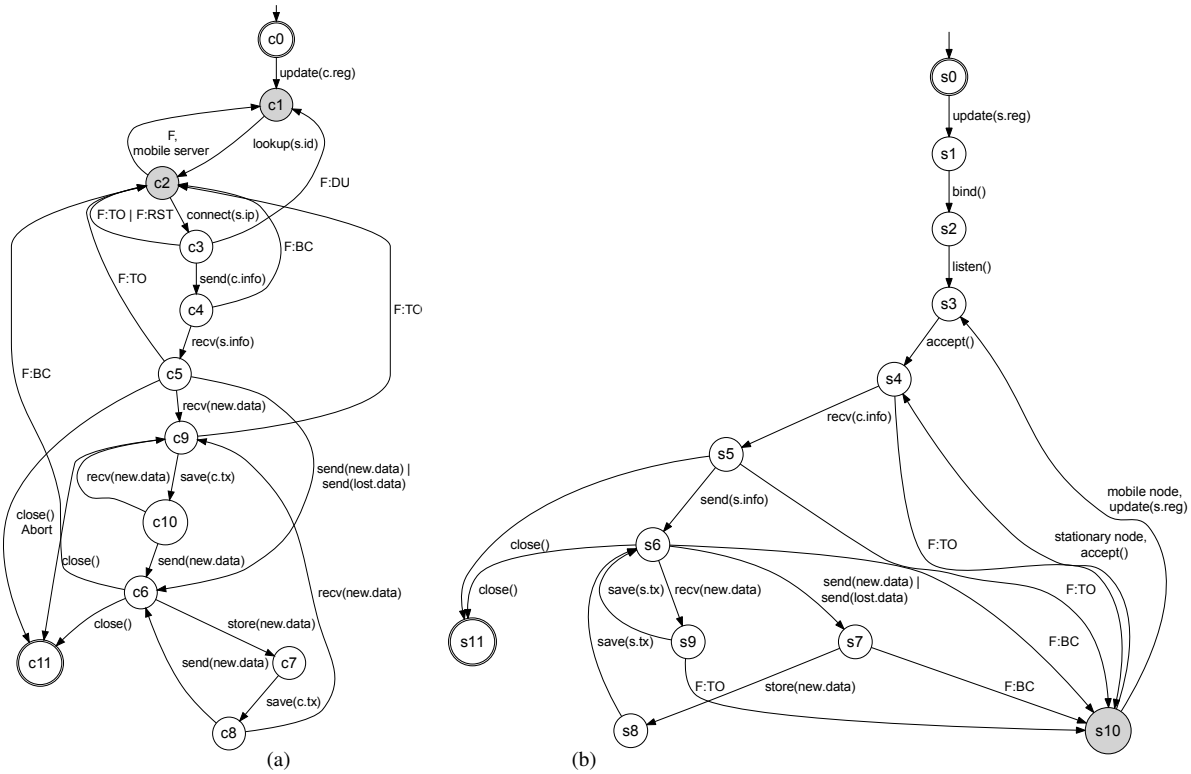


Figure 1. Finite state machines for mobility-aware applications: (a) client, and (b) server.

status by counting the received bytes (c_9 to c_{10}). The client can receive more messages (c_{10} to c_9), send new messages (from c_{10} or from c_8), or finish the transmission (from c_9 or from c_6).

At the server side a similar approach is used. Once connected, the server receives a new message (s_6 to s_9) from the client and saves its transmission context by counting the received bytes (s_9 to s_6). The server can also send new messages to the client (s_6 to s_7) and save its transmission context by copying the sent message in its ancillary buffer and counting the sent bytes (s_7, s_8, s_6). The server is also able to finish the transmission by closing the connection and reaching the final state s_{11}.

C. Disruption Detection

Modifications in the routing table (e.g.,, adding new IPs or routes) and connection timeout break established TCP sessions. On the attempt of sending a new message on a broken TCP socket, an error is returned after calling the send function. In the client, the handling is conducted in the transition from state c_5 to c_6, in which the error `F:BC` (*broken connection*) leads the system to state c_2 again. In the server, the same failure leads to the erroneous state s_{10}.

In the receiver, unread bytes can remain in the socket receive buffer. Thus, the receiver will not crash immediately after the disruption. Instead, the connection becomes half-open, so that the receiver stays in deadlock waiting for messages from a dead sender. To quickly detect a broken connection in the receiver's side, parameters of the TCP Keepalive is reduced as much as possible [14]. Thus, the receiver detects disruption with connection timeouts `F:TO` according to the transitions c_5, c_9, c_2 and s_6, s_9.

D. Resuming communication from broken connections

After a failure, when the transition leads to state c_2, a re-connection has to be established to resume the communication in the client. In order to do so, the server must be resynchronized and prepared to accept a new re-connection from the client again. Then, the server must be waiting in state s_3. By executing the session handshake, client and server are aware about the current transmission context of each other, as shown in the transitions from c_2, c_3, c_4, c_5 and from s_3, s_4, s_5, s_6. This resumes the end-to-end communication into a new TCP connection. The session is preserved by authenticating the peer using the key material and the host identifier. Thus, the server relates the incoming re-connection to an existent session.

E. Restoring Transmission Status

After the TCP connection re-establishment and the handshake between the peers, the sender restores the transmission context by resending the lost data recovered from the ancillary buffer. To determine the amount of lost data, both client and server must know the amount of bytes that was sent and

received by each other until to the disruption. To do so, the peers can mark transmission checkpoints by counting the bytes that are sent and received successfully.

We suggest the use of in-band signalling between the peers. Session control information, including transmission checkpoints, are exchanged over the data connection, as we propose with handshake between client and server with transitions from c_3, c_4, c_5 and from s_4, s_5, s_6, respectively.

The sender's amount of data lost in handover is given by:

$$N_{ld} = N_{ls} - N_{rr}, \quad (2)$$

where N_{ls} is the local amount of bytes sent by the node, and N_{rr} is the amount of bytes received by the remote peer.

The node is able to recover the lost data from the ancillary buffer and, then, resend them to the opposite node with the function `resend(lost.data)` in the transition from c_5 to c_6 and from s_6 to s_7.

While resending, however, a disruption might eventually cause a connection failure again. Since the lost data are already saved, only new messages are copied into the ancillary buffer. The model provides the same support for handling this disruption as described previously. In that case, the remote peer can receive part of the lost data that the sender tried to resend. Although this sending is incomplete, with the next handshake the peers recalculate the new amount of lost data, which denotes the remaining of lost data that has to be sent. This strategy provides integrity of the data transmission in the whole session duration and brings reliability to the Application Layer of the mobile nodes.

F. Mobility scenarios and Location Management

In a *Single Jump* scenario, the server usually runs on a stationary host to which the client re-connects. The mobility becomes a more complex problem when the server node also changes its location. In this scenario, called *Double Jump*, location management is required. Some entity must provide the current location of mobile server nodes. Besides, there is the possibility of moving both nodes simultaneously, which increases the chance of race conditions.

Race conditions are experienced when the client tries to re-connect to the server when it is not prepared to accept connections. This occurs when: i) the server visits a new network and the client does not know about the new server's location; ii) the client detects the disruption before the server, so that the client starts the re-connection in advance.

Mobility protocols can leverage Dynamic DNS for mobile node location [7]. *Rendezvous Servers* [8][10] have also been applied to location management. However, we suggest the use of distributed mechanisms to store node locations. A Distributed Hash Table (DHT) of general purpose, such as OpenDHT [15], is able to provide the necessary scalability to manage mobile node location. A DHT provides storage based on key-value semantic, as well as a simple put-get interface. Thus, to query the peer's location, a node uses the peer's host identifier as a key for searching in the DHT. To update location, a node puts its location register under its host identifier in the DHT.

A node is aware of the role it plays in a session. Thus, when the server is migrated, if it runs on a mobile node, it puts its location register in the DHT (s_{10}, s_3). Otherwise, it just waits for re-connections from the client (s_{10}, s_4), without performing a location update. In the client, a connection failure arises when the server is migrated. The client is aware of the role the server plays due to the control flags exchanged in the handshake. Then it can query the current server's location in the DHT with the server's host identifier. If the client attempts to re-connect to the outdated server's location, the connection fails, since the destination is unreachable (`F:DU`). Then, the client stays in the loop from c_2, c_3, c_1 until the race condition is overcome.

III. HANDOVER DELAY MODEL

To evaluate mobility-aware applications we applied the mathematical model proposed by Mohanty and Akyildiz [11] and extended by Shah *et al.* [12].

As discussed in [12], the end-to-end handover delay is the sum of the following delays:

$$T_{handover} = T_{L2} + T_{IP} + T_{sig} + T_d, \quad (3)$$

where they represent, respectively, the delay of: Link Layer connectivity, IP address acquisition, handover control messages signalling, and one-way data packet transmission in new network.

The critical delay is T_{sig}. It varies according to the approach used by the mobility protocol. Next, we discuss the model used in [11] and [12], which is based on the number of control messages exchanged between mobile node (MN) and correspondent node (CN) to complete end-to-end handover.

A. Handover signalling delay as function of packet loss probability

The handover operation model of TCP-Migrate [7] discussed by [11] can be applied to the majority of end-to-end mobility management protocols [12]. As in [16], the model is based on the delay of TCP connection establishment as function of end-to-end packet loss probability and of retransmission timeout. In mobility scenarios, then, the model represents the delay of connection set-up or migration between MN and CN in new visited network. It is calculated as function of end-to-end packet loss probability, number of handover signalling messages exchanged between the peers and their number of retries, and the retransmission timeout.

The delays of handover signalling completion for end-to-end mobility protocols of two [12] and three control messages [11], respectively, are

$$L_{sig}(i,j) = RTT + (2^i + 2^j - 2)RTO, \quad (4)$$

 ISBN: 978-1-61208-279-0

and,

$$L_{sig}(i,j,k) = 1.5RTT + (2^i + 2^j + 2^k - 3)RTO, \quad (5)$$

where i, j, and k are the number of unsuccessful tries for handover control messages, RTT is the round trip time of the message, and RTO is the initial retransmission timeout for the TCP connection.

Using Equation (4) and (5), we express the overall delay of handover signalling completion for a end-to-end mobility protocol of N_m handover control messages as

$$L_{sig}(A) = \frac{N_m}{2} RTT + \left[\left(\sum_{i=1}^{N_m} 2^{a_i} \right) - N_m \right] RTO, \quad (6)$$

where $A = (a_1 \ a_2 \ a_3 \ \ldots \ a_{N_m})$ is the number of unsuccessful tries for each handover control message.

The end-to-end packet loss probability between MN and CN discussed by [11] is

$$p = 1 - (1 - FER)^f (1 - p_{wr}), \quad (7)$$

where FER is the link layer frame error rate; $f = \frac{L_p}{L_f}$ is the number of link layer frames per packet, where L_p is the length of the packet, and L_f is the length of the Link Layer frame; and p_{wr} is the packet loss probability in wired link part.

From the TCP latency model in [16], the authors in [12] and [11] discuss the probabilities of handover signalling completion for protocols of two and three control messages, respectively, by expressing

$$P_{sig}(i,j) = p_1^i (1-p_1)\ p_2^j (1-p_2), \quad (8)$$
$$P_{sig}(i,j,k) = p_1^i (1-p_1)\ p_2^j (1-p_2)\ p_3^k (1-p_3), \quad (9)$$

where p_1, p_2, and p_3 are the packet loss probabilities calculated using Equation (7) for each individual handover control message with its respective i, j, and k number of unsuccessful tries. These equations represent the probabilities of end-to-end mobility protocols to complete handover after the exchange of i unsuccessful tries for the first control message, followed by the first succeeded message, followed by j unsuccessful tries for the second control message, followed by the second succeeded message, and so on.

Assuming the number of unsuccessful retries in A, for a mobility protocol of N_m control messages, we express the overall probability of handover signalling completion as

$$P_{sig}(A) = \prod_{i=1}^{N_m} p_i^{a_i} (1 - p_i), \quad (10)$$

where p_i is the end-to-end packet loss probability calculated using Equation (7) of ith control message with its respective a_i number of unsuccessful tries.

The average of the handover signalling messages delay for a protocol of $N_m = 3$ control messages, such as TCP-Migrate, is [11]:

$$E[T_{sig}] = \sum_{i=0}^{N_r-1} \sum_{j=0}^{N_r-1} \sum_{k=0}^{N_r-1} P_{sig}(i,j,k)\ L_{sig}(i,j,k), \quad (11)$$

where N_r is the number of retransmissions before giving up and abort the connection establishment of TCP-Migrate. As discussed in [12], N_r is used as the maximum allowed number of retries for handover control messages.

Using the overall delay and probability of Equation (6) and (10), we generalize the average of the handover signalling messages delay for any N_m-message protocol as

$$E[T_{sig}] = \sum_{A=1}^{N_A} P_{sig}(A_i)\ L_{sig}(A_i), \quad (12)$$

where N_A is the number of all the combination of A taking into account N_m and $N_r - 1$.

B. Handover signalling delay for mobility-aware applications

According to the state machines for mobility-aware application, the handover signalling delay at the active opener (mobile client) in single jump scenario is

$$T_{sig_{SJ}} = T_{TCP} + T_{hs} + T_{ld}, \quad (13)$$

where T_{TCP} is the new TCP connection establishment delay between MN and CN (from c_2 to c_3), T_{hs} is the protocol handshake delay which is used for end-to-end synchronization and/or authentication (from c_3 to c_5), and T_{ld} is the lost data retransmission delay (from c_5 to c_6). These delays are calculated using Equation (12), each one according to the number of control messages N_m exchanged between MN and CN.

The handover signalling delay at the active opener in double jump scenario is

$$T_{sig_{DJ}} = T_{lookup} + T_{TCP} + T_{hs} + T_{ld}, \quad (14)$$

where T_{lookup} is the client's delay to lookup server's location. T_{TCP}, T_{hs}, and T_{ld} are defined in Equation (13).

To lookup location register, the requester sends a request to the DHT and waits response from it. The DHT is accessed using either Sun RPC over TCP or XML RPC over HTTP [15]. Assuming that the delay to get a register is the sum of the time to establish a TCP connection with the DHT and the message round trip time (lookup request and its reply), the registration location lookup delay can be expressed as $T_{lookup} = T_{TCP} + T_{req} + T_{rep}$.

 ISBN: 978-1-61208-279-0

However, as discussed in [11], the round trip time for a lookup request can be expressed as twice the one-way end-to-end delay transportation. Thus,

$$T_{lookup} \quad = \quad T_{TCP} + 2(D + t_w), \qquad (15)$$

where D is the Link Layer access delay, and t_w is the one-way delay in the wired network between the new Base-Station/Access-Point and the remote node (DHT).

IV. Simulation Results

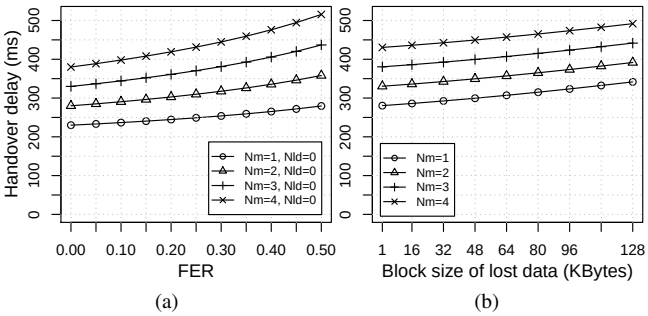
We evaluated the cost of handovers for the mobile active opener by using the described handover delay model in both single and double jumps; i.e., the handover latencies for the client to leave the state c_1 (in double jump) or c_2 (in single jump) and, then, reach the state c_5 (if there are no lost data to resend) or reach the state c_9 (after resending the lost data). Both communicating sides accomplish equivalent operations to handle F, therefore, the cost of handover is considered the same for both client and server. We implemented the handover delay model using the GNU software environment for statistical computing provided by R[17].

A. Simulation Parameters

In most application protocols the connection is considered to be established when SYN/ACK packet arrives at the active opener [16]. This is because immediately after sending ACK in the third packet of the three-way handshake, the active opener sends a data segment to the passive opener that contains the redundant ACK [16]. Then, we assume that TCP connection is considered to be established by the client with 2 control messages. Thus, we set $N_m = 2$ in Equation (12) to calculate T_{TCP}.

The number of control messages exchanged during the handover signalling depends on the synchronization and/or authentication used by the mobility protocol. The mobile node can confirm its new location by simply sending a single control message to the correspondent node. SIP [8] requires at least exchanging two control messages using re-INVITE and 200 OK confirmation. The FSM suggests exchanging two control messages to provide end-to-end synchronization and unilateral authentication. TCP-Migrate [7] requires exchanging three control packets with the three-way handshake needed to migrate TCP connections. When secure connection re-establishment is a requirement, a four-way handshake can provide mutual challenge-response authentication and end-to-end synchronization [18] [3]. We evaluated these different handover signalling strategies for mobility-aware application by varying the values of N_m from 1 to 4 in the calculation of T_{hs} from Equation (12).

To compare these different handover signalling strategies, we assume the values of the handover delay model parameters as in [12]: $T_{L2} = 10$ ms, $T_{IP} = 20$ ms, $T_d = 50$ ms, $RTT = 100$ ms, $RTO = 200$ ms, $pwr = 1e-6$. Since most TCP implementations abort connection establishment attempts after 4-6 failures [16], we assume $N_r = 4$ retries. We set $L_f = 576$ bytes (length of Link Layer frame), regarding the packet size every host must be able to handle on the Internet [19]. The length of the packet $L_p = 40$ bytes for the TCP connection establishment, regarding the size of IP header and TCP header, both with no options. We assume $L_p = 60$ bytes for the length of the handover control message, which is able to accommodate the control information (host identifier, checkpoints, and protocol flags) we described in Section II-A. As in [11], we set the link layer access delay $D = 10$ ms in WLANs, and the one-way delay in the wired network $t_w = 50$ ms.



(a) (b)

Figure 2. Handover delay in single jump scenarios: (a) as function of the frame error rate; (b) as function of the lost data (N_{ld}) in handover.

As argued in [12], although these values are not constant and may vary depending on the network conditions, when the values are changed they affect in the same manner the behaviour of all the mobility strategies evaluated.

B. Handover Latencies

Figures 2 and 3 show handover delay results for $N_m = 1, 2, 3, 4$ strategies in single and double jumps, respectively. Figures 2(a) and 3(a) show delays as function of FER when there are no lost data in handovers, i.e., $N_{ld} = 0$, hence, $T_{ld} = 0.0$. In both scenarios, handover delays increase as the FER increases, which is natural. When FER gets high, it increases the probability of dropping packets and, therefore, handover control messages need to be retransmitted N_r times to successful complete the handover. The overall cost of using a handover control message was 62.18 ms in average for both mobility scenarios.

Figures 2(b) and 3(b) show delays as function of the lost data in handovers. We set FER $= 1e-3$ and evaluated the lost data by varying N_{ld} in L_p with multiple blocks of 16 KB up to 128 KB, which is the maximum amount of lost data by a sender. We assume such limit due to the maximum size of 131071 bytes for socket send buffers in GNU/Linux systems with Kernel version above 2.4. We observed that the cost of retransmitting a block of 16 KB of lost data with a single operation `send(lost.data)` increases the handover latency in 7.65 ms.

 ISBN: 978-1-61208-279-0

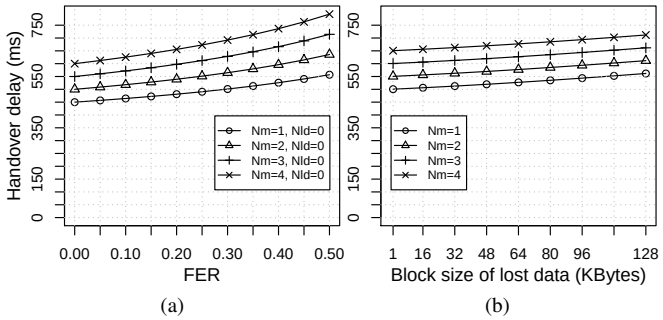


Figure 3. Handover delay in double jump scenarios: (a) as function of the frame error rate; (b) as function of the lost data (N_{ld}) in handover.

In single jumps, which is the common mobility scenario, the mobile client can be moved with no need for location updates. Therefore, the entity for managing location is avoided. This allows reduced handover latencies than the ones in double jumps, as shows Figure 3. Due to the registration location lookup delay (T_{lookup}), in which the client gets the current server's location registration from the DHT in order to re-connect, the handover latencies in double jumps are 42% in average bigger than the ones in single jump; i.e., a mean overhead of 244.37 ms.

We observed that mobility-aware applications provide little and smooth impact when both FER and lost data increase. This behaviour can be an interesting advantage in dynamic scenarios, such VCNs, where devices are of high mobility and packet loss rates are prone to be higher than the ones in mobility scenarios focused on the mobile user.

Comparing results in [12], $N_m = 2$ strategy (as primarily suggest our FSMs) provides handover latencies very close to the kernel space based mobility solutions, such as TCP-R, TCP-Migrate and HIP. In relation to these solutions, an overhead lower than 100 ms is the weight of the burden on handling mobility at the user address space of mobile nodes. However, this performance degradation is derisive when taking into account the benefits of providing easy deployment and maintenance of full support for mobility, which can be implemented as a single software service.

V. Conclusion

In this paper, we described the general purpose mobility-aware applications by means of Finite State Machines capable of providing transitions between erroneous and safe states. These transitions are abstract operations based on the Socket semantic, which are necessary to resume consistently and lossless the end-to-end communications broken with mobility. The presented abstraction is useful in support for developing mobility solutions as software services at the end nodes involved in TCP sessions, with no relying on home networks or agents to intermediate the communication. Applying the analytical model for handover delay discussed in [11] and [12], simulation results show good performance and smooth impact in situations of high packet loss rates and of lost data.

VI. Acknowledgements

We thank INCT-SEC for funding this work by means of the agencies CNPq and FAPESP.

References

[1] C. E. Perkins, "IP Mobility Support for IPv4, Revised," in *IETF RFC 5944*, November 2010, pp. 1–100.

[2] D. Johnson, C. Perkins, and J. Arkko, "Mobility Support in IPv6," in *IETF RFC 6275*, July 2011, pp. 1–169.

[3] B. Y. L. Kimura, H. C. Guardia, and E. S. Moreira, "Disruption-Tolerant Session for Seamless Mobility," in *WCNC 2012: Proceeding of the 2012 IEEE Wireless Communications and Networking Conference*, 2012, pp. 2412 –2417.

[4] B. Y. L. Kimura and H. C. Guardia, "TIPS: Wrapping the Sockets API for Seamless IP Mobility," in *SAC'08: Proceedings of the 23rd Annual ACM symposium on Applied computing*, vol. 3, March 2008, pp. 1940–1945.

[5] W. M. Eddy, "At what layer does mobility belong?" *IEEE Communications Magazine*, vol. 42, no. 10, pp. 155–159, October 2004.

[6] D. Funato, K. Yasuda, and H. Tokuda, "TCP-R: TCP mobility support for continuous operation," in *ICNP'97: Proceedings of the IEEE International Conference on Network Protocols*, October 1997, pp. 229 – 236.

[7] A. C. Snoeren and H. Balakrishnan, "An end-to-end approach to host mobility," in *MobiCom'00: Proceedings of the 6th ACM Annual International Conference on Mobile computing and Networking*, August 2000, pp. 155–166.

[8] J. Rosenberg, et al., "SIP: Session Initiation Protocol," in *IETF RFC 3261*, June 2002, pp. 1 – 269.

[9] V. C. Zandy and B. P. Miller, "Reliable Network Connections," in *MobiCom'02: Proceedings of the 8th ACM Annual International Conference on Mobile computing and Networking*, September 2002, pp. 95–106.

[10] R. Moskowitz, P. Nicander, and P. Jokela, "Host Identity Protocol," in *IETF RFC 5201*, April 2008, pp. 1 – 104.

[11] S. Mohanty and I. F. Akyildiz, "Performance analysis of handoff techniques based on mobile ip, tcp-migrate, and sip," *IEEE Transactions on Mobile Computing*, vol. 6, no. 7, pp. 731–747, July 2007.

[12] P. A. Shah, M. Yousaf, A. Qayyum, and H. B. Hasbullah, "Performance comparison of end-to-end mobility management protocols for tcp," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1657 – 1673, 2012.

[13] D. M'Raihi, J. Rydell, S. Bajaj, S. Machani, and D. Naccache, "OCRA: OATH Challenge-Response Algorithm," in *IETF RFC 6287*, June 2011, pp. 1 – 38.

[14] B. Y. L. Kimura, R. S. Yokoyama, R. R. F. Lopes, H. C. Guardia, and E. S. Moreira, "Prototyping applications to handle connection disruptions in end-to-end host mobility," in *WONS 2010: the Seventh IEEE International Conference on Wireless On-demand Network Systems and Services*, 2010, pp. 1–8.

[15] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu, "OpenDHT: a public DHT service and its uses," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 4, pp. 73–84, August 2005.

[16] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *INFOCOM 2000: Proceedings of the Nineteenth IEEE Annual Joint Conference of the IEEE Computer and Communications Societies*, March 2000, pp. 1742 – 1751.

[17] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2008. [Online]. Available: http://www.R-project.org

[18] B. Y. Kimura, R. S. Yokoyama, H. C. Guardia, and E. S. Moreira, "Secure connection re-establishment for session-based ip mobility," in *CBSEC 2012: Proceeding of the 2nd Brazilian Conference on Critical Embedded Systems*, 2012, pp. 58–63.

[19] J. Postel, "Internet Protocol," in *IETF RFC 791*, September 1981, pp. 1 – 45.

 ISBN: 978-1-61208-279-0