Mobile TCP socket for secure applications

Vu Truong Thanh, Yoshiyori Urano
Graduate School of Global Information and Telecommunication Studies, Waseda University, Japan
thanhvtr@aoni.waseda.jp, urano@waseda.jp

Abstract— **A TCP session uses IP addresses (+ IP port) of both end points as identifiers. Therefore when a mobile handover to a new AP that belong to a different subnet/domain, the IP address will changes and ongoing TCP connections are reset. Several approaches have been proposed to solve this problem, and one of which was to modified the TCP/IP stack to update the changes of the IP address for the ongoing connections [5] [6]. However, these proposals causes unnecessary processing when TCP is used in applications which have already employed some kinds of security measures, such as SIP. This paper proposes the Mobi Socket, which specifically supports TCP mobility for intrinsic secure applications without unnecessary overhead.**

Keywords— **TCP mobility, secure applications**

I. Introduction

TCP/IP was developed when all network nodes were stationary, and connection to a network is through cable, therefore it is unthinkable that a node will move to another subnet while accessing to the Internet, and the IP address of an end host is assumed to stay unchanged while a computer is running. As a result, IP address (together with IP port) is used as identifiers for TCP session, and the TCP layer at the end host maintains TCP control blocks (TCBs), which hold the IP addresses and IP ports of both ends for each TCP connection to find the right socket for each datagram it receives from the IP layer.

But with the introduction of wireless access technologies such as Wi-Fi, it is possible for a mobile node to handover to a new AP that belong to a different subnet/domain while actively connecting to the Internet. This causes an IP address change, and for current implementation of TCP/IP stack, all ongoing TCP connections are reset. This will cause problem for long-live TCP sessions.

There are two general approaches to solve the problem of changing host IP address for TCP session. The first one uses the split-connection approach, which introduces a fix middle agent between the mobile host (MH) and correspondent host (CH) [4]. The connection between CH and MH is broken into two parts, the fixed part between the CH and the agent remains unchanged regardless of the position of the MH, and the TCP connection between the agent and the MH will be re-established whenever the MH handovers to a new address. In this sense only the TCP at the MH is affected, while at the CH the TCP session is not disturbed. The problems of these approaches are non- transparent end-to-end operation of TCP session, as well as the requirement of new infrastructure entities (the middle agent) and triangle overhead.

The other approach modify the TCP stack so that when the mobile host changes the connection to the internet, the TCP stacks at both ends preserver the TCP connection and update the TCBs with the new IP address at both ends accordingly.

In [5], when the MH changes its location, the proposal in [5] introduced new states to the TCP specification. When the address of MH changes, MH and CN will exchange information and update the new IP address accordingly. Both sides will prepare in advance a share-secret, and use this share-secret to authenticate each other during the update process.

The proposal in [6] employs a similar concept, but instead of changing the TCP stack, it uses kernel extensions and a user-level redirect daemon process (this was the design of the prototype in BSD). The daemon process will monitor the wireless network interface for changes of IP address, and if one is detected, the daemon at the MH will inform the counterpart at the CH to update the new IP address together. To secure the update process from malicious acts, MH and CH also need a share-secret in advance.

The problem with [5] is that both sides has to perform additional works to exchange a share-secret in advance, regardless of whether the MH will actually performs the handover to a new Access Point (AP) or not, or whether the TCP session lives long enough to experience a handover. The proposal in [6] relieves this matter by initiating the preparation process only if the TCP connection exists longer than a threshold. However, if the MH does not perform a handover, then all of the preparations for the long-live connections are wasted.

One more problems with [5] and [6] is that processing the share-secret for authentication will requires a lot of processing, which in turn consumes battery power at the MH. If many TCP connections are used (such as if the user constantly browsing the Internet) then battery life will be shortened considerably. Moreover, both [5] and [6] are not applicable in the case where both ends perform handover simultaneously.

In the next parts of this paper, we propose a new type of socket called the TCP MobiSocket, that remains connected even if the concerned IP address changes. It works like normal TCP socket, but does not get reset when the IP address at either end changes, and with an additional updateTCB() member function to update the TCBs with the new address. All of the security issues that are required to secure the update of the new address will be handled by the calling applications. This new socket is dedicated to support mobile TCP session for intrinsically secure application, without all the above mentions problems of [5] and [6].

II. Design of the MobiSocket

Logically, there are two phases when mobile device handover. First, the Network interface/card disconnects from the old AP. Then it connects to the new AP. In traditional TCP stack, the network stack at the MH will close all TCP connections in cleaning-up activities, as well as reset the TCBs during these phases.

On the other hand, all of the ongoing MobiSocket will remain in ESTABLISHED state, when the IP address changes, waiting to be updated by the application.

We design a new socket that allows the application to update the change of PoA at both end hosts. The socket is designed based on the following assumptions/requirements:

- There are cases when the TCP connection needs explicit handling before communicating using the new IP address (Re-establishment/update of Security Association for VPN, sending the PATH message of RSVP for QoS, etc…)
- The application takes care of security activities regarding the update of the new address. The reasons for this are (1) if the connection needs to be secure, the applications have already shared some kind of security, and (2) if the connection is not important to the extend that it requires a shared security association between both end host, then it might not important enough to be hacked by others.
- Compatibility with applications using legacy TCP/IP stack is desired to promote deployment. It means that in the case the other end does not support the features of mobile socket, connection will work according to that of legacy TCP specification.
- Being able to provide handover of TCP session between different network interfaces of the same mobile device. The requirement is that not only IP address but change of TCP port also must be supported, because the same port of the other network interface might be in used by a different application at the time of the request for handover.

The application which uses the *MobiSocket* will call the MobiSocket's `updateTCB()` member function to update the TCB with the new destination address.

To satisfy the above requirements, the mobile socket will provide the following APIs to the applications:

■ **`acvMobi(socket_id)`**

`socket_id: the handler of the socket`

➢ The application will call this function to explicitly activate the mobile feature of the socket

➢ If this function is not called, then the MobiSocket will work like normal TCP socket

➢ When this function is called, the TCP connection will not be abolished if the concerned wireless interface changes to a new IP address

■ **`updateTCB(socket_id, direction, newIPaddress, newPort)`**

`socket_id: handler of the socket`

`direction: update the source or the destination address`

`newIPaddress, newPort: the new IP address and new port to update to TCB/PCB (TCP Control Block/Protocol Control Block). If the port is 0 then keep the existing port value`

➢ The application will call this function to update the TCB/PCB (TCP Control Block/Protocol Control Block) with the new source/destination address and port

➢ The MobiSocket will start a new congestion control algorithm called the mobile congestion control

■ **`copyTCB(new_socket_id, old_socket_id)`**

`old_socket_id: handler of the old socket`

`new_socket_id: handler of the new socket`

➢ The application will call this function to update the TCB/PCB (TCP Control Block/Protocol Control Block) of the newly created socket with the information of the old socket. This is used when the application want to handover from old interface to new interface.

➢ This function will copy all information of the old socket (include current states, CWND, AWND, RTO etc…, except the source IP address and source Port) to that of the new socket, and then delete the old socket without sending FIN to the other end (i.e., application at the CH).

Apart from the above two new APIs/member functions, the MobiSocket also introduces two new message, the *AddChange and AddConfirm.*

The *AddChange* contains (1) A shared token between Mobile Host (MH) and Correspondent Host (CH), (2) the old IP address and the new IP address encrypted by the private key of the MH, (3) the new port address and (4) The old IP address of the MH in plain-text

The *AddConfirm* contains (1) the shared token between MH and CH, (2) the new IP address encrypted by the private key of the CH

If the two messages above are implemented as TCP header options, then these header options must be sent to the applications, but currently there is no mechanism to perform such action. Therefore, it might be better to send this as OOB (out-of-band) data using the TCP Urgent Pointer.

III. Working Procedure of the MobiSocket

Let's consider the use of *MobiSocket* for a SIP application. Suppose that a TCP connection is established between MH and CH (the thick, solid line), which have established a SIP session through the SIP server. The *MobiSocket* will work as follows (see figure 1):

- First the application creates the TCP socket for the SIP session, and calls the `acvMobi()` to activate the mobile feature for the socket
- In step ①, the MH moves from Subnet 1 to Subnet 2, and in the process its address change from IPaddress1 to IPaddress2
- In step ②, the SIP application at the will call the `updateTCB()` function to replace IPaddress1 with IPaddress2 at the TCB table. Then it issues a SIP INFO message to ask CH update the new IP address of the MH.
- Upon receiving this INFO message in step ③, the SIP application at the CH will authenticate the message using SIP security associations, and all the `updateTCB()` function to replace IPaddress1 with IPaddress2 at the TCB table.
- Then in step ④ the SIP application at the CH will send back the INFO message back the the MH to confirm the change of address. Note that both INFO messages may contain other parameters of the concerned TCP session, such as new window size, MSS etc…
- CH and MH will start sending data using the TCP connection when they receives the INFO message from the other end, and they will start receiving data after they send the INFO message to the other end.

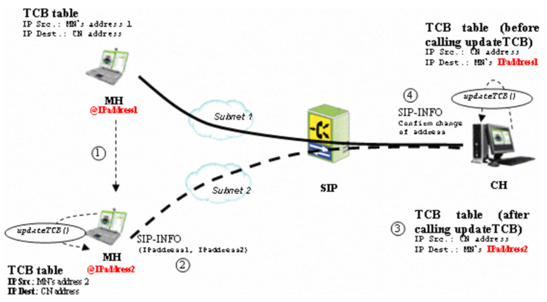

Figure 1. Working procedure of the MobiSocket

IV. Discussion and Optimization

The merits of the *MobiSocket* are:

- Inherit intrinsic security feature of SIP
  - ❑ Less processing overhead for security issues (conserve power)
- Depending on the security requirements, the application can decide whether to allow the handover of TCP connection
  - ❑ More suitable for application with strict security requirements
- Still work when both ends handover simultaneously
  - ❑ Reach-ability through SIP Registration functionality

We can further optimize the operation of the *MobiSocket* as follows:

- When the MN receives the INFO message from CN, both ends might already time out (due to handover, **NOT** due to congestion), so even if the TCB is updated, no data exchange is possible until the time out is over (can be very long). We can provide a new function to reset the timer after the updateTCB() function, which is the resetTimer(socket_id). This function will reset the TCP socket to the state as if it has just received a data/ACK packet from the other machine
- Furthermore, if SIP proxy is used, then normally the MN has to finish re-Registration with the SIP proxy first before it can send SIP INFO message to the other end. This creates further delay for the TCP session. To solve this, we note that the MN and CN can share public key with each other during the initial INVITE process, therefore after the MN handover to a new IP address, it can use the public key of the CN to send the SIP INFO message to the CN right away. However, this solution cannot be used if both ends handover simultaneously (therefore they do not know the IP address of each other), in this case they must contact through SIP proxy server (after the re-Registration process)

V. Conclusion

In this paper we propose the MobiSocket to support TCP mobility for secure application such as SIP. This socket causes no overhead if handover does not take place like previous proposal, and moreover it still works when both side handover simultaneously.

In this socket, there is no need for per-TCP connection authentication, because the authentication is left to application. Depending on the real situation, the application can also control whether to keep the TCP session or not, which is more appropriate for application which is applied with other application level constrains such as security and QoS policy …

In the future, we would like to carry out the implementation of the *MobiSocket* to confirm the design of the system, as well as to measure the delay and throughput parameter when the resetTimer() function is (1) called and (2) not called, and compare the results with that of [5] and [6]. We also would like to measure the delay in the case of SIP application, when we send the INFO message before and after re-Registration, as well as when two end hosts handoff together

We also plan to update the proposal in [1] with this new MobiSocket.

References

[1] Vu Truong Thanh, Yoshiyori Urano, "*Agent based LLMA handover scheme for SIP communication – The case for UDP traffic*", The 11th International Conference on Advanced Communication Technology (ICACT), Feb. 2009

[2] C. Perkins, "*IP Mobility Support for IPv4*", Request for Comments: 3344, IETF, August 2002

[3] Huei-Wen Ferng et. al, "*A SIP-Based Mobility Management Architecture Supporting TCP with Handover Optimization*", Proc. of Vehicular Technology Conference, pp. 1224-1228, Apr. 2007

[4] Milind Buddhikot et. al, "*MobileNAT: a new technique for mobility across heterogeneous address spaces*", Proc. the 1st ACM international workshop on Wireless mobile applications and services on WLAN hotspot, pp. 75-84, Sept., 2003

[5] FUNATO D., "*TCP-R: TCP mobility support for continuous operation*", Proc. IEEE International Conference on Network Protocols, pp. 229 -236 ,Oct. 1997

[6] Vassilis Prevelakis and Sotiris Ioannidis, "*Preserving TCP Connections Across Host Address Changes*", Lecture Notes in Computer Science, Springer Berlin / Heidelberg, pp. 299-310 Oct., 2006

[7] Rosenberg, et. al., "*Session Initiation Protocol*", Request for Comments: 3261, IETF June 2002

[8] D. Yon et. al, "*TCP-Based Media Transport in the Session Description Protocol (SDP)*", Request for Comments: 4145, IETF, September 2005