A Hybrid Cross Layer Architecture For Wireless Protocol Stacks

Zhijiang Chang, Georgi Gaydadjiev
Computer Engineering Laboratory
Delft University of Technology, Mekelweg 4, 2628 CD Delft, the Netherlands
email: [zhijiangchang, georgi]@ce.et.tudelft.nl
Telephone: +31 15 278 6177

***Abstract*—Many architectures to support multiple cross-layer optimizations have been proposed. Most of them can be categorized as either signaling-based or function-call based. The signaling approaches use messages to propagate the information in the protocol stack, while the function-call uses Application Programming Interface (API) for direct access to variables. The signaling approach complies with the layered protocol stack architecture but introduces latency in the information propagation and additionally increases the packet size. The function-call approaches are more efficient but highly operating system dependent. Even a slight protocol modification requires the function-call middleware to be updated. We previously proposed an Infrastructure for Cross-layer Designs Interaction (ICDI) based only on the signaling approach. In this paper, we first provide in-depth analysis of the two main schemes. Thereafter we propose a hybrid version of the ICDI architecture that combines the advantages of both approaches. According to our simulation results, the packet size overhead is reduced by around 75% (in bytes) and the propagation latency is reduced by more than 60%. The proposed hybrid system remains operating system independent.**

I. Introduction

The success of Internet is mainly determined by its layered architecture [1]. A particular layer can be easily replaced or modified while the layer's interface remains unchanged. Therefore, other layers are not influenced by these changes. As for the mobile ad hoc network (MANET) context, the current TCP/IP layered architecture is not suitable. This is because many problems, such as high bit error rate on the wireless link were not considered when the TCP/IP protocol stack was introduced. A way to cope with this is by using cross-layer (CL) designs [2]. CL designs break the layered architecture by using the information from one layer to adjust the actions of another layer, e.g. upper layers that adapt to the changing wireless link situation using information from MAC and physical (PHY) layers.

A CL architecture that supports multiple CL designs should consider the tradeoff between efficiency and modularity. The modularity of any protocol stack means that protocols are operating system (OS) independent and functional independent from each other. The efficiency is measured in term of the information propagation latency and the packet overhead introduced by the architecture. There are mainly two categories of architectures that support CL designs, namely the signaling and the function-call approaches. The signaling approaches favor modularity by using messages to propagate information. Therefore, the modularity is high and the efficiency is low because: 1) the information needs to go through the protocol stack, which increases propagation latency; 2) overhead in term of additional bytes of CL information is introduced to normal packets. The function-call approach, on the other hand, favors efficiency by using APIs to directly access the physical memory location of a variable. The information also does not need to go through the protocol stack. This approach, however, is highly OS dependent because every system has its own unique API. In addition, different versions of the same OS may also have different APIs. Therefore, the middleware of function-call based CL architectures must be frequently updated along with other systems components. This is not practical in nowadays multi-OS, multi-vendor environment. Furthermore, the function-call approach also needs to deal with read-write conflicts because of the direct memory access style.

We previously proposed an Infrastructure for CL Designs Interaction (ICDI) [3]. The architecture allowed multiple CL designs to interact by strictly using the signaling approach for propagation of CL information. An Interaction InterFace (IIF) was introduced to each layer to simplify the encapsulation and de-encapsulation of CL information. After an in-depth analysis of the function-call and signaling approaches, we found possible improvements to reduce the packet overhead and propagation latency of the original signaling-based ICDI while keeping the architecture OS independent. In this paper, we present a novel Cross-Layer InterFace (CLIF) component to replace the IIF in the original architecture. The CLIF uses function-call style features e.g. storing variables using shared memory. In addition, the CLIF is further optimized by using hashing to organize the CL information. The performance of the individual CL designs is also improved in some cases when our novel CLIF is introduced in ICDI.

The main contributions of this paper are:

1. In-depth analysis of the advantages and disadvantages of both, function-call and signaling based approaches for CL information management;
2. A novel Cross Layer InterFace (CLIF) element that uses shared memory for information organization and access while keeping the overall architecture OS-independent;

3. Experimental results to validate the improvements due to our hybrid architecture (the packet overhead is reduced by more than 70% and the propagation latency is reduced by more than 60%).

This paper is organized as follows. We analyze the signaling and function-call based approaches in sections II and III respectively. In section IV we briefly introduce the ICDI architecture and describe the novel CLIF element that replaces the original IIF. The simulation results that evaluate the improvements of our proposal in terms of latency and packets overhead are reported in section V. We finally conclude the discussion in section VI.

II. Signaling Approach

The fundamental difference between function-call and signaling approaches is that signaling approach uses messaging to deliver information among different entities. Every entity only needs to know the packet format (packet header) of its peer. For protocols in different layers, the communication is achieved by adding and removing packet header. In this section, we use the SNMP [4] and the MobileMAN [5] to explain the advantages and disadvantages of signaling approach for CL information architectures.

A. SNMP

The Simple Network Management Protocol (SNMP) [4] is not designed for CL interaction, but for network management systems to monitor network-attached devices. SNMP, however, delivers CL information because the sources of the information are the IP layer and below, while the destination is the application layer. Here we use SNMP to analyze the advantages and disadvantages of signaling approach for CL interaction.

SNMP forms part of the internet protocol suite as defined by the Internet Engineering Task Force (IETF). The SNMP uses GET, SET, and $TRAP$ primitives. The GET or $TRAP$ primitives are used to inform the status (value) of a network variable. The SET primitive is used to update or control a variable or a system. Also, SNMP does not define what information a managed system should offer. Instead, SNMP uses an extensible design, where the available information is defined in a dedicated database called Management Information Base (MIB). These characteristics are very suitable for CL information management. The current structure of SNMP, however, still has the following problems when it is directly used for CL information management:

- The SNMP is an application layer protocol. Other protocols must use the standard TCP/IP protocol stack to access the information in MIB. Consequently more information propagation latency and packet overhead are introduced.
- The MIB is depicted as a tree hierarchy. It is very powerful to manage information but too complicated and not necessary for CL information management.
- The MIB structure is suitable for centralized data management in one protocol. The CL designs, on the other hand, are decentralized in many protocols of different layers.
- The interface between SNMP and other protocols is still not defined because currently SNMP is only used for network management. It is not used by other protocols for any other purpose.

B. Dedicated signaling architecture

The Mobileman [5] provides a solution that breaks the basic layered architecture. The Mobileman architecture adds another stack component called Network Status. This component is a repository provided for network information sharing among the layers. The Mobileman recommends replacing the standard protocol layers with a redesigned network status-oriented protocol architecture. This, however, would lead to increasing implementation and maintenance efforts. Furthermore, the protocols need modification if the Network Status is enhanced. Efficiency would be reduced and complexity would be increased in such cases.

C. Advantages and Disadvantages

The advantages of signaling approach are:

- Simplicity is achieved because the protocol only handles its own header, nothing else.
- The packet header formats are defined by international standardization organizations. All industrial and academic parties can simply connect to the network with the knowledge of the header format.
- Independency is high when using the signaling approach. The signaling approach provides a unified solution for the inter-node and the intra-node interactions because they all handle the packet headers n the same way.

The disadvantage of the signaling approaches is lower efficiency compared to function-call approach. By propagating the information using packets, overhead is introduced because the system needs more memory to store the additional information. In order to provide up-to-date information, signaling approaches may frequently send information using packets and consequently introduce more overhead. Another disadvantage is propagation latency caused by layer-by-layer information propagation. The time complexity analysis of the signaling approach is provided in [3].

III. Function-call approaches

The function-call based solutions establish direct connection between layers by providing APIs to access all required information. This section uses two examples, the ECLAIR and XIAN architectures, to explain the strength and weakness of this approach.

A. ECLAIR

The ECLAIR [6] architecture belongs to the function-call category. This architecture prohibits adaptation loops by exhaustive listing all the functions that may be called by CL designs and providing registration scheme to control the function calls. Because the functions are different in many operating systems, this architecture is operating system (OS) dependent. CL architecture should also provide mechanisms for multiple designs to interact. The ECLAIR architecture does not clearly state this issue or provide any solution.

Another problem of ECLAIR is that all the CL optimizations are regarded as stand-alone system component. Therefore, all the CL optimizations are applications, instead of a part of the protocol that they are optimizing. This is not practical in real version-based software releases such as the operating systems. The CL optimizations are normally a piece of patchable software to the original protocol. After the patch is applied, the protocol will be able to adjust its behavior according to the CL information.

B. XIAN

Another API based solution XIAN is presented in [7]. The authors listed the available MAC layer APIs to provide CL information for the network, transportation and application layers. This architecture only provides bi-directional interaction between MAC and an upper layer. The interactions between IP and TCP, or between TCP and application, are impossible. This significantly limits the capacity of the XIAN architecture. Like the ECLAIR architecture, the XIAN proposal is also OS dependent. The authors actually only provided a solution for Linux which is not the most widely used OS for mobile terminals.

C. Advantages and Disadvantages

The advantage of function-call based solution is high efficiency. The function-call approach is more efficient than the signaling approach because:

- Function-call approaches directly accesses memory to read/write variables. No additional overhead in the packets is introduced when propagating the information.
- Function calls do not need to go through the layered protocol stack, which means the function-call approach has less information propagation latency than the signaling approach.
- No intermediate entity, such as the intermediate layers in the signaling approach, is involved in the function-call process.

As mentioned in the previous section, the most important feature of protocol stack is that when the implementation of the layer changes, the interface is still unchanged. The function-call approach, however, breaks this structure by allowing direct API access to the variables. This makes the function call approach very implementation-related and OS-dependent. This problem cannot be solved even if the middleware of the function-call based architectures exhaustively including all APIs. When the protocols are updated and the names of the APIs are changed, the middleware, the tuning layer of ECLAIR for example, still requires update.

As shown in figure 1, a real network entity that includes applications, OS core and network interface card (NIC) involves the software/hardware from many vendors. Listing all possible APIs is impossible. Furthermore, the MAC and PHY layer information may be directly generated by hardware so that no API is available.

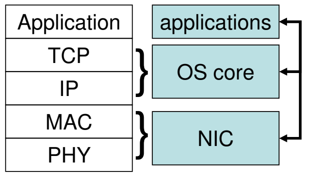

Fig. 1. TCP/IP protocol stack and its implementation in real systems

Inter-node CL interaction using APIs, with Remote Procedure Call (RPC) for example, are not efficient. RPC is application layer protocol. This means the overheads of all layers are introduced even for the interaction that only involves lower layers. Furthermore, most intermediate nodes such as routers do not have an application layer. This makes the API based inter-node solution impossible.

Another weakness of all function-call approaches is that the write-write conflicts exist when multiple entities try to directly write the same physical memory in a real-time system.

IV. Hybrid Architecture for CL information Management

We envision that there is a possibility to combine the advantages of both approaches discussed in the previous chapters. The simplicity and the OS-independence of the signaling architectures can be combined with the efficiency of their function-call counterparts by using direct memory accesses in a single integrated approach.

An Infrastructure for CL Designs Interaction (ICDI) was previously proposed in [3] shown in figure 2. This architecture uses layer-by-layer signaling indicated as packets flow in the figure to propagate CL information. The CL information is present only in the protocol (not shown in this figure) that provides it. The middleware handles the priority of multiple CL designs. Interaction InterFaces (IIF) are added to each layer to simplify the encapsulation and de-encapsulation of CL information as. Compared to the function-call approaches, this architecture has longer information propagation latency and additional overheads due to the distributed variables storage and the propagation approach.

In order to reduce the propagation latency and the packet overhead of the original ICDI, we propose a novel CLIF element that replaces the IIF. The CL information is stored in a hash table. Shared memory is used to access this information from different layers. The signaling based propagation is used for the inter-node interactions and intra-node interactions that may lead to write-write conflicts.

A. Novel Cross Layer Interface component

The CLIF is a component introduced to each protocol layer that replaces the original IIF. The CLIF is responsible to:

- decode the CL information from packets when packets arrive;

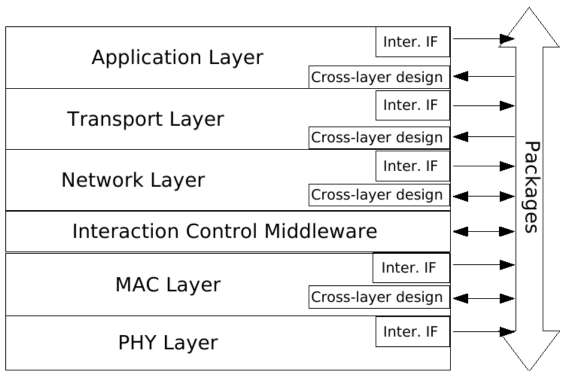

Fig. 2. the Infrastructure for Cross-layer Design Interaction

- provide information to core protocol in XML format;
- propagate the CL information to other layers when a packet is sent out.

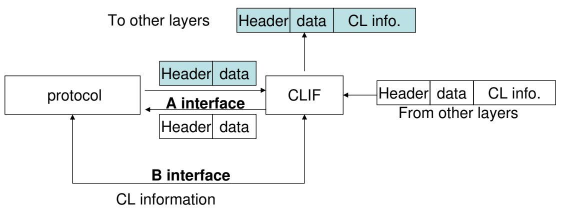

Fig. 3. CL interface and packet handling

The CLIF provides two interfaces for the protocols as shown in figure 3. The “A Interface” is the interface that transfers the standard packet. Therefore, a protocol that cannot handle the CL designs is still functioning in this architecture. The “B Interface” is available only to CL aware protocols and is used to exchange the CL information with the CLIF. This interface uses simple ASCII encoded commands, e.g. $action, parameters...$. The CLIF supports the following commands related to variable handling:

- SET, *variable_name, value*: The SET command is used by protocols to add or update a variable in the CLIF.
- GET, *variable_name*: The GET command helps the CLIF to retrieve the value of a variable from a protocol. The complete value format is returned to a buffer. The protocol then needs to cast the value to the desired data type before using it.
- $UPDATE$, *variable_name, value*: The $UPDATE$ command is used by the CLIF to request a protocol to update its parameter.
- $CANCEL$, *variable_name*: The $CANCEL$ command is used by protocols to cancel the registration of a variable. If the $CANCEL$ operation is successful, the variable is removed from the CLIF so that no other protocol can access it.

The protocols use SET to update the information stored in the CLIFs. Please note that there is a single copy of each variable that can be accessed by all CLIFs (this will be explained later). If the protocol requesting the update is not the owner of the variable, an $UPDATE$ action is used by the CLIF to synchronize the local value in the protocol that owns it. $UPDATE$ is also used when inter-node CL designs need to write variables in other nodes.

In order to enable such interactions between the CLIFs and protocols, the protocols need to support the above APIs as they need to support other protocols’ header formats. The following is the difference between the CLIF API and the API in the functional-call approach. The function-call approaches require the protocols to provide APIs, while CLIF provides standard APIs to protocols. The APIs provided by different protocol implementations are not standardized, but the APIs provided by CLIF (SET, GET, $UPDATE$, $CANCEL$) are universal.

Therefore, the following is the complete procedure of incoming packet handling with the CLIF:

1. The CLIF handles the CL information before the packet is handled by the destination protocol.
2. After removing the CL information in the packet, the CLIF forwards the packet to the destination protocol.
3. The CLIF sends the GET command to request the CL information from a protocol.
4. The protocol uses the SET and $CANCEL$ functions via “B Interface” to access and manage the CL information in the CLIF.
5. When a protocol sends out a packet, the CLIF attaches the CL information that is to be forwarded. Therefore, the standard protocols are not responsible for handling the CL information. Instead, they access the CL information from the CLIF of its layer.
6. The CLIF uses $UPDATE$ to synchronize the value in its shared memory and the protocol that owns the variable (the owner protocol).

B. Compatibility

In order to cooperate and coexist with network entities that do not support CL optimizations, the CL architecture should allow the protocols to operate without knowing the existence of any CL information. The CL architecture needs to be notified if additional information is attached to the normal packet. Appropriate pre-handling de-encapsulates the additional information before the packet is handled by the standard protocol. We use the unused bits in the protocol headers to indicate if CL information is attached to normal packets. Then the novel CLIF detaches the information from packets if present.

MAC and PHY layers provide most of the CL information that is involved in the CL designs. Therefore, by default, the packets from MAC layer are attached with CL information. In the IP header, one bit in “flag” is reserved. The IP header provides notification by setting this flag. There are still four reserved bits in the TCP header, after the Explicit Congestion Notification (ECN) [8] uses two of the six original reserved bits. The IP header also has one bit reserved in the "flag" field. These bits can be used to notify the CLIF if the packet contains additional CL information. The application layer has dedicated methods to negotiate parameters like QoS configuration for lower layers. This layer generally only reads CL information from the CLIF. Consequently, no flag in application headers is needed to indication CL information.

C. CLIF with hash table

In order to improve the search speed and simplify the storage of CL information, we use a hash table to store CL information in the CLIF. The name of the layer where the variable comes from is the hashing key as shown in figure 4. This is because the variables matched to the same key belong to the same layer. Therefore, the protocol can efficiently update all its own CL information.

The time complexity of inserting an item is $O(1)$. The time complexity of searching an item is in the worst case $O(n)$, where "n" is the number of CL variables in this layer.

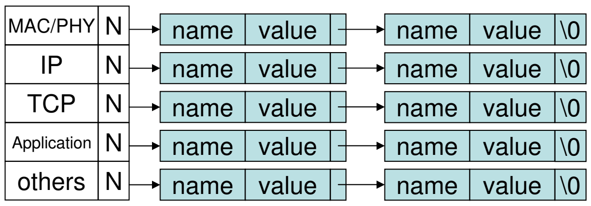

Fig. 4. The hash table to store CL information in CLIF

D. Shared Memory for Information Synchronization

Instead of storing a complete copy of CL information from all layers, the CLIF only stores the CL information from its own layer. The CLIFs use the memory address reference to each other to access the CL information that does not belong to its layer. According to the CLIF data structure shown in figure 3, the position of variables can be precisely calculated given the layer and name of the variable. By referencing the addresses instead of copying the values, the synchronization of variables is achieved. Figure 5 also clearly shows that this implementation detail is hidden from the protocols that use the CL information.

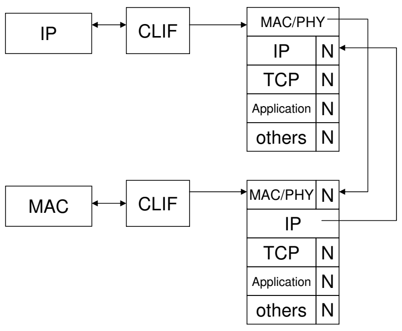

Fig. 5. The shared memory of CLIF hash tables in MAC and network layers

The challenge of shared memory approach is the write-write conflicts. The write-write conflict is introduced when multiple protocols update a value at the same time. In order to remove such conflicts, in our proposal the signaling approach is used when a protocol needs to update a parameter that is not in its own layer. The message is sent to the CLIF of the destination layer. The CLIF then updates the value in shared memory and uses $UPDATE$ action to synchronize the value in the protocol. Therefore, only the CLIF API from the same layer as the parameter can update the parameter in shared memory. The write-write conflict is thus avoided.

In order to accelerate the speed of fetching variables, the CLIF maintains the name list of the stored variable. Therefore, the protocols can use the search function provided by the CLIF to directly access the physical address of the variable. The following algorithm is used by protocols to fetch a variable from the shared memory.

```
// This is the definition of CL information
// stored in the CLIF.
struct CLInformation {
    name;
    value;
    owner;
    last_update_timestamp;
}
FUNCTION FetchInformation (Name)
{
    // data is the reference to CL information
    CLInformation *data;
    // Search for the index "N" of information in shared
    // memory using the CLIF function.
    N = searchByName(Name);
    // Calculate the start address of the CL information
    address = StartAddress + N * SizeOf(CLInformation);
    // Fetch data, cast it to the desired type.
    data = (CLInformation *)address;
    return data;
}
```

E. Packet Overhead Consideration

In the original signaling-based only ICDI, complete information about a variable including the layer, the protocol, the name and the value is propagated along with normal packets. With the proposed shared memory access mechanism, this additional information is significantly reduced. Instead of transporting the complete information description, only maintenance information is attached in most cases. The complete CL information is only propagated along with normal packets when a protocol from other layer needs to update a CL variable. The packet overheads introduced by the hybrid architecture include:

- registration of the CL designs;
- broadcast that notifies all layers that a CL parameter is available or unavailable (canceled);
- write request in case that the parameter and the protocol that updates it are not in the same layer.

V. Validation and Results

In order to validate our novel CLIF, we implement the CLIF in the Network Simulator 2 (ns-2) version 2.28 [9]. The new implementation is based on the original implementation of ICDI with the following improvements:

- the original IIF is replaced by the CLIF using the hashing and shared memory mechanism;
- The system time is used to evaluate the propagation latency inside a node;

The same simulation scenarios to validate of the original ICDI are used in this paper in order to compare the performance of the original signaling-based only architecture and the improved hybrid architecture with novel CLIF. We use the following network scenario:

- mobility model: random waypoint [10];
- number of nodes: 50;
- area: 500m $\times$ 500m;
- node speed: 0 to 20m/s;
- data sources: 25 FTP on TCP, and 10 CBR on UDP;
- test duration: 600s, pause time is the time when the nodes are stationary;

Ad hoc On-demand Distance Vector (AODV) [11] and Optimized Link State Routing (OLSR) [12] are used in order to investigate the behavior of both proactive and reactive routings. The NS-2 OLSR patch is provided in [13]. We still use the three CL optimizations that validate the original architecture in [3]. The details of the three CL optimizations were explained in [3].

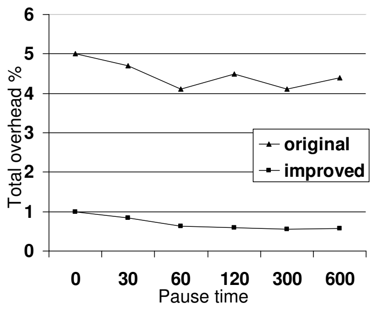

Fig. 6. Overhead improvement

Figure 6 shows the reduction of packet overhead in the new design. The overheads of both the original and the improved systems are proportional to the number of packets. Consequently, the pause time only has minor effect on the results. The packet overhead reduction is mainly contributed by the smaller size of additional information in the new design.

The system time is used to compare the propagation latency of the original and the improved systems. The latency is the period from the time when a variable is requested, to the time when the value of variable is fetched. Because the absolute propagation latency is strongly related to the memory and the CPU used for the simulation, the normalized values of the latency (value of the original ICDI as 1) are compared. We find the improved design is averagely 70% faster to deliver the information value to their destinations. This validates that the novel CLIF with shared memory access mechanism introduces less latency than the original ICDI using signaling propagation.

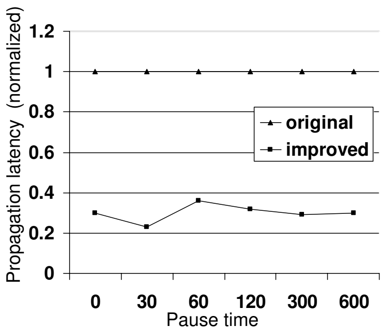

Fig. 7. Normalized propagation latency comparison

The $up - to - date$ CL information is defined as: At time t, a protocol in layer d reads the value of the variable V (V_{td}) from the CLIF; The value of V in the layer s that provides this information is V_{ts}; The information is up-to-date if $V_{ts} = V_{td}$. We further define the information $correctness$ as the ratio of queries that fetch up-to-date CL information to the total number of such queries.

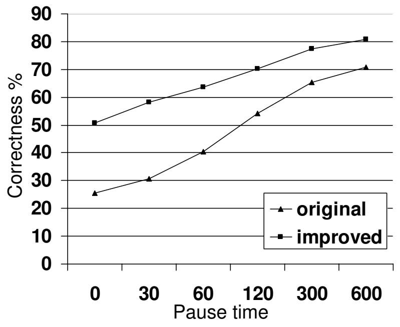

Fig. 8. Information correctness comparison

Figure 8 shows the information correctness of the original signaling-based and new hybrid architectures. The hybrid architecture always provides better results because correctness increases when the information propagation latency is reduced. However, it is impossible to guarantee 100% correctness because any real system has computation delay as well as propagation delay. Neither the signaling nor the function-call approach can completely avoid these delays.

With reduced information propagation latency, the protocols can access more CL information that is up-to-date. Consequently, the successful delivery of packets, measured in terms of throughput percentage, is improved as shown in figure 9 and 10. The improvements mostly happen when the MAC/PHY layer CL information is changing rapidly due to fast changing topology (small pause time). When the topology becomes more stationary, the CL information changes less frequently. Consequently, less improvement is achieved. The end-to-end packet delay is also slightly improved (around 1%) using the new hybrid architecture.

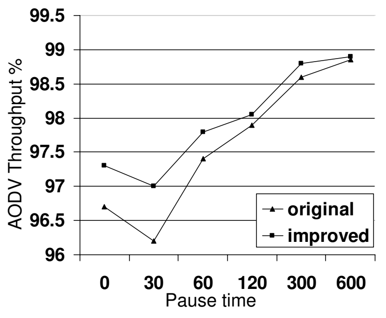

Fig. 9. AODV throughput percentage comparison

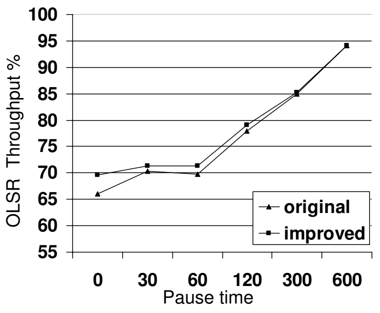

Fig. 10. OLSR throughput percentage comparison

VI. Conclusion

This paper analyzed the advantages and disadvantages of the existing CL architectures based on the function-call and the signaling approaches. Then we proposed a hybrid CL architecture using a novel cross-layer interface (CLIF) element that combined the advantages of both the signaling approach's simplicity and OS independency, and the function-call approach's efficiency. This architecture is an improved version of a previously proposed CL proposal. The information propagation latency and packet overhead were reduced by 75% and 60% respectively in the new design. The proposed CLIF used a hash table to efficiently organize the CL information stored in the memory. The shared memory mechanism further reduced the total required memory space. The write-write conflicts were avoided by providing separate access mechanisms to a particular variable for the owner layer and the other protocol layers. The synchronization of the variable value in the CLIF and that in the owner protocol was also achieved using the shared memory mechanism.

Acknowledgment

This work is supported by the Dutch Government as part of the decree on subsidies for investments in the knowledge infrastructure (Bsik) program. This work is done within the Micro Satellite (MISAT) project.

References

[1] I. Chiamtac, M. Conti, and J. Liu, “Mobile ad hoc networking, imperatives and challenges,” in *Ad Hoc Networks, vol 1, no. 1*, July 2003.

[2] V.Kawadia and P. Kumar, “A cautionary perspective on cross layer design,” in *IEEE Wireless Commun., vol. 12, no.1.* IEEE, feb. 2005, pp. 3–11.

[3] Z. Chang, G. N. Gaydadjiev, and S. Vassiliadis, “Infrastructure for cross-layer designs interaction,” in *the 16th IEEE International Conference on Computer Communications and Networks (IC3N)*, August 2007.

[4] D. P. J. Case, R. Mundy and B. Steward, “Rfc 3410 introduction and applicability statements for internet standard management framework,” 2002.

[5] M. Conti, G. Maselli, G. Turi, and S. Giordano, “Cross-layering in mobile ad hoc network design,” in *Computer, Volume 37, Issue 2,*, 2004, pp. 48–51.

[6] V. Raisinghani and S. Iyer, “Architecting protocol stack optimizations on mobile devices,” in *Communication System Software and Middleware, 2006. Comsware 2006. First International Conference on*, Jan. 2006, pp. 1–10.

[7] H. Aiache, V. Conan, J. Leguay, and M. Levy, “Xian: Cross-layer interface for wireless ad hoc networks,” in *Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2006.

[8] N. W. Group, “Rfc 3168 the addition of explicit congestion notification (ecn) to ip,” in *Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, 2006.

[9] “The network simulator - ns-2,” in *http://www.isi.edu/nsnam/ns/*.

[10] T. Camp, J. Boleng, and V. Davies, “A survey of mobility models for ad hoc network research,” in *Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking Research, Trends and Applications, volume 2, number 5*, 2002, pp. 483–502.

[11] C. Perkins, E. Belding-Royer, and S. Das, “Ad hoc on-demand distance vector (aodv) routing,” in *Internet Engineering Task Force (IETF) draft*, July 2003.

[12] T. Clausen and P. Jacquet, “Optimized link state routing protocol (olsr),” in *Internet Engineering Task Force (IETF) draft*, October 2003.

[13] F. J. Ros, “University of murcia um-olsr documentation,” in *http://masimum.dif.um.es/um-olsr/html/index.html*, March 2005.