Abstract— In this paper we present a research testbed being developed at K.U.Leuven, Belgium, aiming at providing a flexible and convenient testing infrastructure for researchers working on mobility issues over IP networks. The testbed is built on general-purpose PCs, which makes it suitable for most laboratories. Its user-level modularity makes it easy for both developers and users to configure and extend even for unusual mobility scenarios and new mobility techniques. Movement is realized in a way that makes mobility experiment easy to be carried out and automated.

Keywords- mobility; testbed; Mobile IP; modular router

I. INTRODUCTION

It is commonly believed that an important feature of the future Internet is the support for mobility. This is especially true when more and more small devices such as PDAs and handsets also start to access the Internet. Supporting mobility here means a mobile device is allowed to move around and change its point of attachment to the Internet, while its ongoing communications are not disturbed. The main reason why normal IP networks do not support mobility is that IP address here serves two purposes at the same time: it is used for both identification and routing. When the mobile device is outside its home network domain, it will not be reachable on its home IP address any more; but obtaining a new IP address from the visited network domain results in a different identity. Active research is carried out to solve this mobility problem. A brief survey of the major network and application layer approaches can be found in [1].

During the research of mobility techniques, to complement the results from simulations, research infrastructures are also desired to obtain experimental results in a more real-life environment. Unfortunately, setting up a research infrastructure or testbed that meets all the specifications of the intended experiment is not an easy task and can be very time consuming, especially when it is used for new techniques or unusual scenarios that are not yet supported by any existing system. Adaptation, modification and extension will be necessary in these cases, which makes a lot of mobility testbeds unsuitable.

In this paper we try to target these problems by introducing user-level modularity into the mobility testbed. With fine-grained modularity in the user space, users now have a much stronger control over the testbed. The advantages such as flexibility, extendibility and reusability that come with a modular architecture are now also shared by the users, which greatly speeds up the process of building a tailored testbed with particular mobility features.

In section 2, we will first give a brief overview of the current third layer mobility techniques, which are essential to a mobility testbed. In section 3 we will discuss how we can construct a mobility testbed and show the advantages of a modular structure compared to the other alternatives. Detailed design considerations will be presented in section 4. In this section we will also describe some other features of our testbed that are useful for mobility experiments. In section 5 we will outline the future work. Section 6 concludes the paper.

II. IP MOBILITY TECHNIQUES

There are quite some mechanisms developed at the network layer to provide mobility support for IP networks. For the widely deployed IPv4 networks, the most important one is Mobile IPv4 [9]. It is normally used for global mobility. Most other network layer mechanisms supplement Mobile IPv4 to provide better service on a more local scale. Mobile IPv4 (abbreviated as Mobile IP from now on) itself also has some enhancements and extensions to tackle its weaknesses.

A. Mobile IP

Mobile IP solves the problem of mobility by dividing the two roles of a conventional IP address into two different IP addresses and binding them together. The basic way Mobile IP works is shown in Fig. 1. The original IP address a node has on the network it belongs to (called the node’s Home Network or HN) will only be used to identify the node when it moves to another network (called Foreign Network or FN), and a new IP address it obtains in the FN will be used for correct routing. Since the original IP address (called Home Address or HAddr) never changes, applications using this address to identify the

Figure 1. Mobile IP

We are indebted to the Flemish government, which partly sponsors this research through the generic university basic research program (GBOU) “End-to-End QoS in an IP Based Mobile Network”. 
Mobile Node (MN) will not be disturbed; on the other hand, the new IP address (called Care-of Address or CoA) belongs to the FN, therefore the MN can be reachable again on this CoA. The binding between HAddr and CoA is maintained by a special agent in the HN called Home Agent (HA). Each time the MN moves, a handover happens and the binding in the HA is updated. For IP traffic destined to the MN, the HA will intercept and tunnel them to the MN’s CoA. Another agent in the FN called a Foreign Agent (FA) can assist the MN to obtain its CoA. An MN can work without a FA if it can obtain a CoA through some other mechanisms such as DHCP.

B. Macro mobility and micro mobility

Every time a MN moves, it has to register to the HA to update the binding table. This registration process takes time, and it can cause a long service break when there is a large RTT (Round Trip Time) between the MN and its HA. If the MN changes network frequently, the problem will be even worse. Naturally we can reduce the impact by decreasing the frequency of this lengthy registration process.

This can be achieved by having a bigger domain in the foreign region which is divided into smaller parts, such as the cells shown in Fig. 2. A domain can be a bigger FN or a cluster of FNs. The intra-domain movement is called micro mobility, comparing to the inter-domain one which is called macro mobility. The registration to the HA is limited to the macro mobility case, where a normal Mobile IP handover takes place; in case of micro mobility, the movement is handled locally so that no position update is needed for the HA. There are several different ways to support micro mobility. One of them is called Regional Registration which is a direct extension to Mobile IP.

In Regional Registration [10], adjacent FNs are organized into bigger domains, and the movement within this domain is not registered to the distant HA. Instead it is registered to an agent local to the domain called a Gateway Foreign Agent (GFA). As long as the MN moves within the domain, its position is only known to the GFA and the visited FA. The HA only knows that the MN is attached to that GFA, and tunnels all the packets for that MN to the GFA, which again forwards them to the MN. Only when the MN moves out of the domain, the HA registration is necessary. Since local registrations are much faster, the total handover delay of the MN will decrease.

There are other approaches for micro mobility, such as Cellular IP [11], Handoff-Aware Wireless Access Internet Infrastructure (HAWAII) [13], and Telecommunications-Enhanced Mobile IP (TeleMIP) [12]. Interested readers can find more details about them in the given references.

Currently most techniques listed here already have some implementation available. For popular ones like Mobile IP, several implementations are available from different institutes. A straightforward way to set up a testbed for these mobility techniques would be just using these implementations directly. Unfortunately, this is not possible for every technique, since not all proposed techniques are implemented; implementing them yourself from scratch is possible but painful and time consuming. Even when there are implementations available, they may still lack something you need, as we will see later.

III. MOBILE IP TESTBEDS

In this section we will discuss the way how to build a testbed suitable for research with Mobile IP. We start with various Mobile IP implementations that already exist. Here only open-source software implementations are considered. They run on cheap general-purpose PCs and do not require specialized hardware. And because of their open-source nature, users can modify them for their own needs. These features make them advantageous for research activities.

A. Mobile IP Implementations

The SunLabs Mobile IP software [2], developed by Sun Microsystems, is widely used in the academic society such as in [6] and [7]. It relies on a kernel service called vtunl for tunneling and de-tunneling functions. All the other functions are provided by user-level programs. These programs can turn a computer into a mobility agent or a mobile node.

SunLabs Mobile IP is also used as the basis for some other implementations, such as Dynamics from Helsinki University of Technology [3] and UoB zNOMAD developed by ComNets at the University of Bremen [4]. Dynamics introduces the support for hierarchical Mobile IP; UoB-zNOMAD adds new features like the support for multiple points of attachment and provides a GUI for the software. At Stanford University, the MosquitoNet Mobile Computing Group has developed another Mobile IP implementation which provides enhancements like bi-directional tunneling [5] but only supports co-located CoAs that work without an FA.

All of these implementations comply with standard Mobile IP, and apparently a lot of research on Mobile IP has been carried out or validated in testbeds based on one or several of these implementations as in [6], [7] and [8]. But still, whether these testbeds are sufficient for all the diverse purposes of different mobility experiments is still to be considered.

B. Limitations of Existing Implementations

As stated above, these implementations provide standard compliant functionalities for a Mobile IP network. Because of that, a MN from Dynamics should be able to register to a HA from MosquitoNet and vice versa. On the other hand, each of them also provides some unique features that are not supported by the others. In order to have all the features, you may need to include all these implementations in your testbed, as in [8]. But you can not have all of them running at the same time on the same machines, simply because they are not designed to...
coexist on the same host. Because of this incompatibility problem, you cannot have both Dynamics and UoB-zNOMAD running to support multiple points of attachment of a MN in a hierarchical Mobile IP network. For these scenarios, you will have to port the wanted features from one implementation to another. Since different implementations are designed in different ways, porting code can be very difficult.

Modifying and extending these implementations is not trivial either. Code-level modularity as in SunLabs Mobile IP only helps within each network entity. When a new type of entity is needed, for example the GFA of Regional Registration [10], you will have to code it from scratch even when its major functionalities are quite the same as existing entities. Of course you can port code pieces from old entities to the new ones, but reuse on that level is error-prone and hard to maintain.

Another limitation of these implementations is that they do not have enough flexibility. Working scenarios are limited. Users can only tune the parameters that are allowed to be tuned and use the provided tools for predefined commands. Changing the behavior freely is not envisioned and therefore is not possible without modifications to the source code, even when no new functionalities are involved.

Furthermore, in these implementations changes can only be applied after recompilation. Sometimes rebooting the machine or kernel re-patching is also required. This routine has to be repeated even for the slightest change, which drives the inefficiency caused by the lack of flexibility even further.

C. Modular Structure in a Mobile IP testbed

In the discussion above, we identified the limitations of building Mobile IP testbeds just with existing implementations. They are sufficient if you only need basic mobility support. When the testbed is built for advanced mobility study, these limitations make them unsuitable. We need a mobility testbed with a generalized structure that can accommodate different techniques and is easy to use, modify and extend. It should have the following features:

1. The architecture should be general, so current and future mobility techniques can be supported in the same environment and share all the features provided by it. Of course, each technique can be enabled or disabled individually according to the user’s need.

2. It should be easy to modify and extend. The place for modification should be easy to locate and isolate. When introducing extensions, changes to the original system should be limited, since they can easily introduce unexpected bugs. At the same time, reusability should be taken into account to avoid unnecessary repeating.

3. The structure should be flexible. Different configurations should be able to be carried out in an easy manner, and the effects should be observable in a timely fashion for the efficiency of large scale experiments. Users should not be bound by the possibilities provided by the affiliated tools.

4. Although we only talked about Mobile IP, the testbed is not designed only for network layer mobility approaches. Since there is a trend in mobility research towards mechanisms across multiple layers, cross-layer techniques should be supportable, too.

We believe a modular architecture helps providing these features. The modularity here not only present in the code level but also in the user level, so that both developers and users can benefit from it. A fine-grained user-level modular architecture encapsulates simple functions into small building blocks and all protocols and sophisticated mechanisms can be realized with them. Common functions are put into general building blocks which can always be reused by different techniques. Technique-specific functions are realized in building blocks that are not for reuse. In this way, modifications are limited within certain building blocks and extensions only need a few new building blocks for functions that are not yet supported.

Because the interconnections among these building blocks are now controlled by the users, the testbed set-up can be changed easily not only by adjusting the parameters of the building blocks but also by rearranging them in different ways. As long as there is no need to change the building blocks themselves, you do not have to reset the whole system.

In the following section, we will describe in more details the design considerations of this modular testbed.

IV. TESTBED DESIGN CONSIDERATIONS

We did not start from scratch to build such a testbed. Instead, we chose the Click Modular Router Project [14] as the basis. Click is a software architecture for building flexible and configurable routers. The building blocks in Click are called elements. Individual elements implement simple router functions like packet classification, queuing, scheduling and interfacing with network devices. Fig. 3 shows a simple network sniffer implemented with Click. It gets packets from one network interface, keeps a copy of them for later use, and pushes the packets out untouched to another interface.

Fig. 3 shows that a Click configuration is a directed graph with interconnected elements at the vertices. Packets flow along the edges of the graph to get processed in the elements. Configurations in Click are written in a simple language and executed separately. It is a declarative language and changes in the configuration can be applied immediately. Interested readers are referred to [14] for more details.

Click provides the user-level modular architecture we are looking for. With the Click language, users can choose which elements to use and how they are interconnected. In this way a complete router with the desired behavior can be easily realized and configured. However, Click by itself is not yet sufficient for building up a mobility testbed, because it does not have the support to any kind of mobility mechanisms.

![Logical Movement Realization](image_url)
In this section we present the design considerations to add mobility support to the Click architecture. There are three main concerns: first, mobility entities do not function in the same way as routers do, thus some assumptions in Click will not hold any more and need to be adapted. Second, there are various mobility techniques and extensions. The way how we support them will affect the extendibility and flexibility of the testbed. Last, we also need some other features that are essential for mobility research but are not yet supported, namely movement realization and experiment automation.

The discussion below uses Mobile IP and its Regional Registration extension as an example, but for other mobility techniques the same principles still apply.

A. Mobility support

In order to maximally keep the modularity of Click, new functions of mobility entities should be carefully analyzed to fit into small processing modules that can be realized with either existing elements or new ones. When creating new elements, we can still increase the reusability by identifying common functions and putting them into generalized elements; the rest that are too specific to be reusable will be put into technique-dependent core elements, as illustrated in Fig. 4. We start the design with the analysis of the protocol.

1) Protocol Analysis

The main tasks of each Mobile IP entity are listed as below:

**Home Agent:** An HA takes care of the packets that are sent to a MN which is away from the HN. It advertises its existence in the HN; it receives registration requests and replies them; it tunnels packets for registered MNs to their CoAs; If the HN is an Ethernet, it also sends proxy ARPs for registered MNs.

**Foreign Agent:** An FA helps a visiting MN remain reachable when it is in the FN. It advertises its existence in the FN; it forwards registration requests from the MN to its HA and the replies from the HA to the MN; it detunnels packets from the HA and passes them to the correct MN.

**Mobile Node:** An MN moves between different network domains. It tries to detect movement and registers to the HA accordingly. When at its HN, it functions as a normal node; when at a FN, it sends all outgoing packets to the gateway or accordingly. When at its HN, it functions as a normal node; and the replies from the HA to the MN; it detunnels packets from the HN and pass them to the correct MN.

From the functionalities listed above, some new functions that are commonly used can already be identified and put into generalized elements. For example all agent advertisements can be implemented with one new element and used for both the HA and FA. The type and content of the advertisement can be specified when using this element. There are also some less-straightforward common functions. For example, when a new registration is accepted, the HA needs a new element to send an unsolicited proxy ARP message to update all the ARP caches in the HN; although not called an proxy ARP, the MN also needs to send out an unsolicited ARP reply to update all the ARP caches in the HN when it comes back home – in the latter case, we can say that the MN sends a proxy ARP message for itself. Because these two processes are practically identical, we put them into a same new element. For those functions like registration message handling that are specific to the Mobile IP protocol, it makes sense to put them into technique-specific core elements. We will talk about them later.

Of course, most basic functions are already realized by existing elements, such as classification and NIC (Network Interface Card) accessing. We can just reuse them in the configuration. Together with the new ones we have made, the complete functionality of Mobile IP can be realized.

2) Core elements

We put only the registration handling functions into the core elements, one for each mobility entity. The reason is that no other techniques than Mobile IP and its extensions will use the same registration mechanism. Although also crucial to Mobile IP and new to Click, functions like tunneling are still less coupled with the protocol and can be reused in other techniques, therefore are realized with generalized elements.

In order to keep the modularity clean, we integrate all the registration records into the core elements. That is to say, opposite to a global database approach, data in our testbed is kept local to where they are created and processed. For example for a HA, when it needs to create/update/delete a registration entry, it will do that locally in the core element. This design complies with the overall design principle of Click, where information is incorporated into the packet forwarding path instead of within shared structures. Following this principle guarantees that these core elements can work together with existing elements.

This design principle is practical for routers since in a router each basic function can work alone and make decisions rather independently according to the content of the packets and its own settings. No run-time information sharing is really needed: a routing table is only needed by the routing element, and a classifier always classifies incoming packets into one of several predefined classes. Static information like IP address and its own settings. No run-time information sharing is really needed: a routing table is only needed by the routing element, and a classifier always classifies incoming packets into one of several predefined classes. Static information like IP address sharing can be passed to the elements as their configurations before the router is online and kept local thereafter.

This design also makes elements easier to use, since they are now complete and independent modules. Relevant data are taken care of by the elements themselves and the interaction between data and processes is now internal to the elements. Of course, these elements can export method interfaces to make the data structures also accessible from outside.

3) Information distribution

As explained above, in Click information is kept local to the elements where it is manipulated. This design is convenient for users and is adequate for routers but not for Mobile IP. In Mobile IP, the way mobility entities treat incoming packets is
more dynamic and depends on the registration entries. For example, a HA only sends proxy ARP replies for registered MNs and tunnels packets for them; for unregistered MNs, it does nothing. This means that some other elements than the core element itself, such as those sending proxy ARP messages and those tunneling IP packets, also need registration info to behave correctly. When the entries are changed, these elements should be updated as well. Therefore, some mechanisms are needed to distribute this information to wherever it is needed.

This is one of the places Click has to be adapted. There are two choices for the core element: an active way or a passive way. In the passive way, whenever an element needs the registration information, the core element will be consulted. This could happen on a packet-by-packet basis and lead to a heavy overhead. The core element can also actively inform all the concerning elements whenever a registration entry changes. This change normally happens at a much lower frequency.

We chose the active way and made the core element also a control element. It distributes registration information actively to all the elements that need it (which are therefore called controlled elements). Every controlled element keeps a local copy of the registration information, but the local copy only contains the part that is needed by the controlled element itself. For example the tunneling element only needs the registered MN's HAddr and CoA, but does not care about the MN's MAC address and the lifetime of the registration. Therefore, it only keeps the HAddr – CoA pair of the MN locally.

4) Chained Structure

From the overview in section 2, we know that there are different extensions to Mobile IP. Generally the extensions only change part of the protocol, so a lot of elements used by Mobile IP can be kept for the extensions. Modification will be limited to a few related elements and new elements are added only when new functions are needed. If the extension changes the registration procedure, a new core element may be needed.

This is a straightforward way to extend the testbed for new techniques, but it does not take backward-compatibility into account. Backward-compatibility means the system should be able to fall back and work in the standard way in case the extension is not supported or not allowed. For example, when a Regional Registration (RR) enabled FA receives a registration request from a RR-enabled MN, it will process the request according to RR and send it to a GFA; if the MN is not RR-enabled, the same FA should process the request according to standard Mobile IP and send it directly to the HA.

Backward-compatibility requires support to the standard Mobile IP in the RR extension. For this purpose we can just have RR functionalities and Mobile IP in the same elements, but this will result in clumsy elements and degraded flexibility. A better way would be to reuse the core elements for Mobile IP in the RR configuration for standard Mobile IP support.

As shown in Fig. 5, backward-compatibility can be realized by arranging the core elements from the base protocol and the extension in two ways: we can either classify the registration messages first and then divert them to the proper core element, or first send all the messages to the RR core element and only pass those that do not comply with RR through to the Mobile IP core element. We call the first structure a tree structure and the second a chained one. We believe the chained structure suits the extension relation better, while the tree one is better for techniques that are more independent from each other.

In the chained structure, we add a set of outputs to the element that map all its inputs. These new outputs are used as the outputs from which input packets can be pushed out again unchanged, if they fail the checking in the element. The chained structure frees the users from creating sophisticated external classifiers to classify subtly different packets. It also introduces priority into the system, where for the same packet one element can always try to process it before other elements do. The latter case is not possible with normal classifiers.

B. Movement Realization

In a mobility testbed, how a MN moves is an essential issue. When a MN moves, it introduces dynamics into the testbed. The mobility dynamics can be realized physically, e.g. changing its physical location and connection to a different network. In a laboratory setup, the MN can switch to another network by connecting its cable to different sockets; in a WLAN (Wireless LAN) environment, the MN can move from the range of one access point (AP) to the range of another.

The movement can also be realized logically: the MN has physical connections to all candidate networks and only one (or several, if needed) of them is active at a time. Each interface has its own link layer implementation but shares upper layers with all the other interfaces. Movement is realized by changing the active connections, and this change can be hidden from the network layer and up.

Both of the two approaches are supported in our testbed, and logical movement has some advantages compared to the physical one. With logical movement, we can mimic a lot of unusual scenarios before real equipment supports them. For example, we can use several interfaces in the testbed to represent a single one in reality, so that this one interface can have access to several networks simultaneously. This is useful, for example, in experiments for fast handoff techniques. On the opposite, we can carry out a logical movement but still use the same interface. By doing this, the routine for movement handling in the upper layers can be generalized and the testbed configuration can be more concise. Furthermore, the logical approach also helps automation. We will discuss it later.

The logical movement is realized with a set of switches that are controlled by a synchronizer. These synchronized switches will always activate the output (or input, in the case of input-switches) of the same index. In the Click configuration, we first connect all interfaces to one of the synchronized switches; wherever there is an interface-dependent process, we make a
copy of the same process for every interface and connect all these copies also to one of the above-mentioned switches, as shown in Fig. 6. In this way, we can enable the first interface by switching all these switches to output 1, and so on. Here the synchronizer controls all the switches. Different algorithms can be employed in the synchronizer to choose which interface is the next one to active. It also accepts commands from users or other elements such as the core element discussed earlier.

C. User Intervention and Automation

In our testbed, modules (elements) are directly accessible to the users, therefore elaborate and user friendly interfaces are desired for each element.

The user interfaces of an element include its configuration string and run-time interaction methods. Configuration strings are used in Click configuration files to initialize all the elements and set values to the internal variables. Run-time interaction methods are called Handlers in Click. They are used to set/get the values of internal data structures or give commands to the elements at run-time. These methods can be utilized by not only users but also other elements. An example is the information distribution mechanism in our testbed: when a core element wants to send information to other elements, as commands on certain elements at certain time. When the program is executed, it will follow this list of commands and call the appropriate Handlers. In this way we can ‘program’ the testbed to carry out experiments automatically in a predefined and repeatable way without human interaction.

The automation of the MN’s movement would be very difficult without the above mentioned logical movement, which is a far more convenient and practical way of repeating movement than moving the MNs physically or connecting/disconnecting cables. In logical movement, changing interfaces can be broken into commands for the synchronizer element and automated with the external programs as explained above.

V. Future Work

The testbed presented in this paper is a work in progress. Development is still going on and more techniques will be introduced. Nevertheless, the testbed already enabled us to start the research with mobility. Our next goal is the support for the other micro-mobility techniques like Cellular IP and HAWAII, so that they can be compared in the same environment.

We will also work on the generalization of the automation process in the testbed. For the moment we use C/C++ to compose automation programs, and need recompilation after every change in the automated task. Having a general automation program that can accept commands as parameters and execute them accordingly is more appropriate. The same generalization work can be done on the core elements as well, so that controlled elements can be assigned to them through configuration strings.

VI. Conclusion

In this paper we presented a testbed that is being developed to provide a flexible infrastructure for research activities on IP network mobility. It is built on a modular software architecture called Click, which is originally designed for building software routers. The user-level modularity provided by the testbed makes it suitable for extensive experiments on new mobility techniques that are still under development. The logical movement and automation features of the testbed help carrying out these experiments in a more efficient way.

VII. References