Modelling application handovers for thin-client mobility

P. Simoens*, L. Deboosere, D. De Winter, F. De Turck+, B. Dhoedt and P. Demeester
Department of Information Technology (INTEC), Ghent University
Gaston Crommenlaan 8 bus 201, B-9050 Gent, Belgium.
{Pieter.Simoens, Lien.Deboosere}@intec.ugent.be
* Research Assistant for the Fund of Scientific Research (F.W.O.-V.,Belgium)
+ Postdoctoral Fellow for the Fund of Scientific Research (F.W.O.-V.,Belgium)

Abstract

Mobile users need lightweight devices with low energy consumption. When applications are executed on remote servers instead of locally on the end-user's device, the weight of the device can be reduced and battery lifetime can be extended. Therefore, thin-clients are ideally suited for mobile users. All calculation logic is removed from the device and the communication with the application server is accomplished by a remote desktop protocol. Reactions to user events can appear on the screen only after a two-way path delay, so application responsiveness can decrease if the user moves further away from his server. To enable ubiquitous thin-client computing with satisfying user responsiveness, the application should make a parallel movement with the mobile user. This paper focusses on the handover between two application servers. A model is presented for the design of a thin-client network that meets the requirements of moving users and the resulting number of handovers is shown. Three mechanisms for the server handover are presented. For each scenario, an estimation is made for the timeframe during which the client does not receive screen updates.

Keywords: thin-client, mobility, application handover

1 Introduction

Mobility has become commonplace in today's networks. Nomadic and mobile users are highly interested in light and easy-to-transport devices. They want to execute their applications fast and reliable on every location worldwide. Battery lifetime is another important issue. Thin-clients can offer a solution to these challenges and are therefore an eligible candidate to enable this ubiquitous computing.
Applications are executed on back-end centralized application servers, instead of locally on the end user's device. Every client event (key strokes, mouse movements) is transmitted over the network towards an application server which executes the commands, renders the appropriate screen updates and returns them to the client. The communication is established by means of a dedicated remote desktop protocol. Some well-known examples are Citrix Independent Computing Architecture and Windows Remote Desktop. All calculation logic is shifted from the end-user towards application servers inside the network. The client only decodes the received screen updates and can thus be made as 'thin' as possible. By removing all processing hardware, the device can be made lighter.
Performing the complex calculations on network servers instead of locally on the end-user's device decreases energy consumption and extends the battery lifetime. Equiping thin-clients with dedicated (lightweight) hardware might further improve the energy balance. Although it should be noted that the continuous (possibly wireless) network

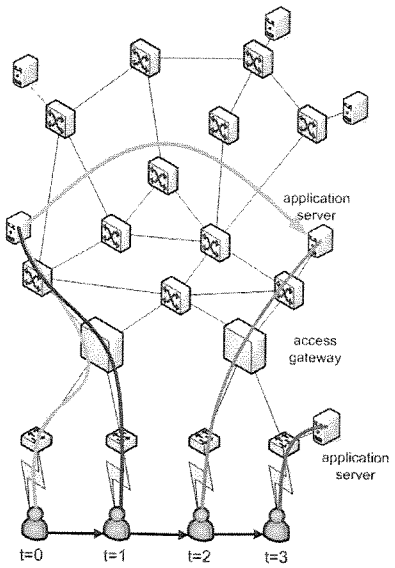


Figure 1: The application migrates along with the mobile user to the closest available application server to guarantee an acceptable delay at all times.

connection with the server has a negative impact on the energy balance, we believe that the total result can be positive by using dedicated hardware and making a well-considered trade-off between the functionality of the user's device and the application servers (i.e. the device can execute lightweight applications locally). The concept of 'Server-Based Computing' brings a myriad of other advantages. The user is released of the burden of daily software and virus definition updates, since this can be offered by the admin team of the application server farm. In corporate environments, central management of the computer park is sufficient and hardware updates become unnecessary, which can result in economic savings. Employees can easily access their regular home office desktop from every affiliate worldwide.

Users expect the network to offer the same functionality and quality from each location worldwide. The thin-client paradigm can inherently suffer from a high latency, since reactions to user events can appear on the screen only after a two-way path delay. If an employee connects from an overseas affiliate to his home office, this delay can become noticeable. A high degree of responsiveness can only be guaranteed by keeping the delay as low as possible. As users travel throughout the world, their applications should make a parallel movement and migrate to the closest available application server. An example use-case for mobile thin-clients is presented in Figure 1. At $t = 0$ the user's application is executed on a server nearby his access gateway (AGW). At $t = 1$, the user has moved and is now connected to another access gateway, but the same server can still fulfil the quality requirements of the user. When the user moves further, this is no longer valid and at $t = 2$, another server will take over the job. The application of the user migrates to the new server, where it is restarted in exactly the same state as before. At $t = 3$, the user reaches his destination. His application is migrated and restarted at the application server of the company he works for.
To enable process migration, sufficiently application servers have to be installed across the network. An overlay management platform will control the handovers between two application servers. This control framework will assign a server to new applications. Existing connections are monitored to guarantee the desired user experience. If the connection quality drops below predefined thresholds, a better application server must be found. The user session is migrated towards this new server and restarted. All of this should happen without the user noticing, or at least with a minimal down-time of the user's applications.

The remainder of this paper is structured as follows. In section 2 we survey related work in the area of process migration and TCP connection handovers. A model for designing a thin-client network is presented in section 3. The number of handovers is calculated for different values of the delay and the results are compared. Section 4 elaborates on the specifics of the handover mechanism. Three possible scenarios are presented and compared. For every scenario the time is calculated during which the client does not receive screen updates. Conclusion and future work are presented in section 5.

2 Related Work

The concept of application (or process) migration between servers was originally developed for load balancing in a server cluster. Some examples are Condor [1] and MOSIX [2]. With these techniques, only processes that do not use inter-process communication or have open network connections can be successfully migrated to another server. Furthermore, migration is not fully transparent, since for e.g. open files are not migrated, and the application at the new server relies on the open files at the original server. More recent process migration techniques like ZAP [3] have solved these drawbacks, by inserting a visualisation layer above the operating system. An application can be fully migrated, including network connections and open files, without leaving any residual state on the original server.
If an application with an open TCP connection must be migrated, the tuple identifying the connection must be updated. ZAP uses a virtual-physical address mapping mechanism [4]. Also MIGSOCK [5] is able to migrate an open connection transparently to a new server. Messages are exchanged to inform the other (not moving) end of the network connection about the connection migration. A comparison of the performance of ZAP and MIGSOCK is made in [6].
The users we want to serve, are mobile. They can move fast, e.g. by train or by car. In [7], research is done to assure connectivity to fast moving users.
In this paper the presence of an effective process migration mechanism is assumed and we focus on the actual handover of the application image between application servers.

3 Application server switching

3.1 Formal Model Description

We developed a model for the design of a thin-client network [8]. The model yields the optimal locations to install sufficiently application servers to meet the delay constraints of the used applications at the lowest possible cost. As stated in the introduction, the user-server path delay should be kept as low as possible. This delay is expressed as a

maximum hop count between the user's AGW and the application server, and may differ for each type of application. The cost of opening a new server farm is separated from the cost of installing a new server in an (existing) server farm. The network is characterized by its set of I nodes and (unidirectional) links:

$$\begin{aligned} N &= \{n_i\} && \text{with } i = 0, 1, ..., I-1 \\ L &= \{(n_i, n_{i'})\} && \text{with } i, i' \in N \end{aligned} \tag{1}$$

The set N is divided into three subsets. The first subset S indicates the possible server locations, the second U the access gateways (AGWs) and the third the intermediate routers. We assume that application servers cannot be installed at the AGWs nor at the intermediate routers.

$$\begin{aligned} S &= \{n_i\} && \text{with } i = 0, 1, ..., X-1 \\ U &= \{n_i\} && \text{with } i = X, X+1, ..., X+U-1 \end{aligned} \tag{2}$$

The parameters α en β denote respectively the cost to install a new server (in an existing server farm) and the cost to open a new server farm. The following objective function is minimized:

$$\alpha \cdot \sum_{i=0}^{X-1} S_i + \beta \cdot \sum_{i=0}^{X-1} loc_i + \gamma \cdot \sum_{t=0}^{T-1} \sum_{j=0}^{J-1} \sum_{k=0}^{K-1} \sum_{(i,i') \in L} a_{ii'jk}(t), \tag{3}$$

with S_i the number of servers at location n_i, loc_i indicates whether a server farm is installed at location n_i or not, and $a_{ii'jk}(t)$ indicates whether link $(n_i, n_{i'})$ is used for routing the traffic corresponding with user j's application k at timestamp t or not. User mobility is simulated by changing the AGW through which they are connected. The delay constraint of the application is counted as the number of hops between the AGW and the server. It is possible that multiple servers can be reached from an AGW without exceeding the maximum allowable delay (hop count). Therefore, a third term is added to equation (3) to assure that in this case the closest server (in terms of hopcount) will be chosen. Parameter γ is used as a weight factor, in order to have the value of equation (3) be mainly determined by its first two terms. The solution has to fulfil several constraints. The delay between the user and the server executing his application may not exceed the maximum allowable delay. User-server traffic should be routed along the links in the network with respect to the link capacity. Other constraints are the fact that a server can only be installed when a server farm is opened at that location; all applications run on exactly one server at every moment (but this server can vary throughout time); and server farm capacity must be sufficient.

3.2 Number of handovers

We applied the model presented above to dimension in a cost-effective way the large city network presented in Figure 2. Two different user scenarios where chosen: train and highway. In the *train* scenario 48 employees, equally distributed among 6 trains, travel to the central station (AGW19 on Figure 2). From there, they spread out to their offices, randomly chosen between AGW20-AGW30. In the second scenario, *highway*, 50 users are travelling by car to their office. They enter randomly the highway around the city and take the nearest exit to their office. Both the travelling direction (clockwise or counterclockwise) and the office are randomly chosen.

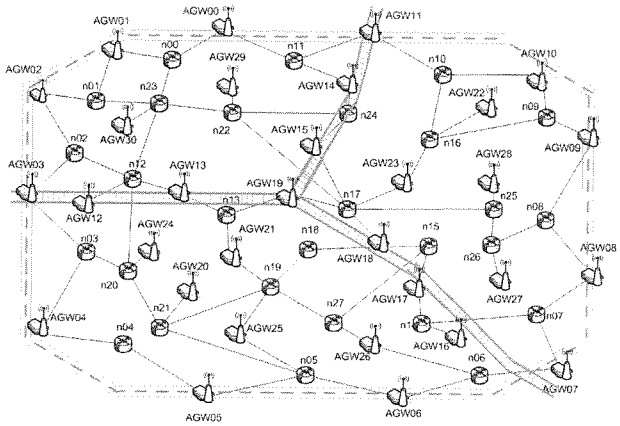
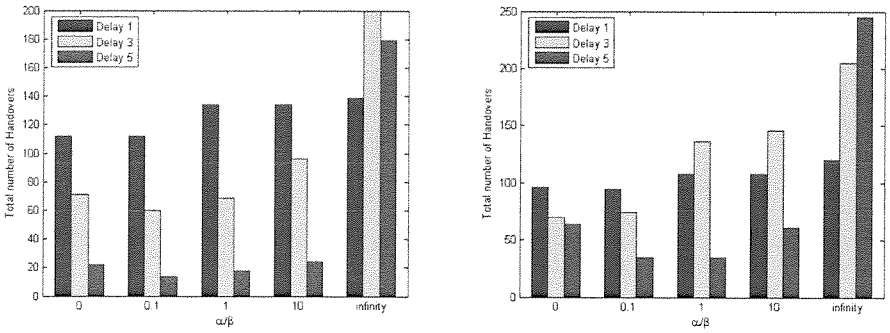


Figure 2: Representation of a city with 3 railroads to the central station and a highway around the city. Users can connect to the network via one of the AGWs. Servers can be installed at every node n_i.

We analysed the simulation results to have an idea of the number of handovers in both scenarios. Figure 3 shows the results as function of α/β. We kept $\gamma = 1$, $\beta = 100.000$ en varied α in equation (3). The infinity case corresponds to $\alpha = 100.000$ and $\beta = 0$. In both scenarios, the global trend in Figure 3 is an increase of the number of handovers with α/β, for every value of the delay parameter. There is always a minimum amount of servers needed to satisfy the demand. The greater α/β, the less the relative cost of opening a new server farm. As a result, equation (3) is minimized by spreading the servers as much as possible (since this does not come with significant extra costs), in order to reduce the third term. This means that every application is executed on a very near application server at all times (see the third term in (3)), although other servers might be within the maximum delay constraint. This incurs more handovers than necessary and explains the increasing trend in Figure 3. The increase is slightly mitigated for a delay of 1 hop and a delay of 5 hops. When only 1 hop is allowed, every AGW to which users are connected, must



(a) Train (b) Highway

Figure 3: Total number of handovers during the whole simulation time (t = 10).

have an application server available within 1 hop. Due to this stringent delay requirement, the number of possible server locations is limited. Regardless of the cost (and hence the value of α/β), the installation of the application servers will not vary very much and hence also the number of handovers will be quite constant. As for a delay of 5 hops, a server can service a lot of the surrounding gateways and the servers can be greatly clustered. Few server farms are opened and due to this clustering, an application will remain on the same server for a longer time. The cost-reducing effect of the clustering in the second term of (3) stays large enough, even when opening new servers becomes relatively cheap. Only with $\beta = 0$ this effect disappears completely, which explains the huge increase of the number of handovers.

The increasing trend is broken by $\alpha = 0$. In that case, installing new servers does not bring extra costs. Again, the same effect will come into play. More servers will be installed to minimize the third term in (3). This comes with an increase of the number of handovers (more frequently switching from application server).

A final remark can be made for a constraint of a 3 hops delay. Normally, fewer handovers are required when the maximum hop count is greater. The cases $\alpha/\beta = 1$, $\alpha/\beta = 10$ (highway scenario) and $\alpha/\beta = \infty$ (both scenarios) seem to be an exception on this trend. We can explain this by referring to the large solution space for an allowable delay of 3 hops. More choices have to be made to select a server out of the eligible server farms, since more servers can be reached from the AGW than when only 1 hop is allowed. The users in the highway scenario make more movements (i.e. a user moves to another AGW) than the users in the train scenario, as can be seen in Table 1. The users in the highway scenario can also connect to other AGWs than in the train scenario. As a result, the locations of the server farms differ between both scenarios. There are more server farms opened in the highway scenario than in the train scenario for the cases where α/β is 1, 10 and ∞. These extra server farms, together with the extra movements of the users, the big solution space and the fact that no penalty is included in the model for a handover, explain why in these special cases, there are more handovers in our model for the 3 hops constraint than for the 1 hop constraint.

Table 1: User movements and handovers for $\alpha/\beta = 1$ and delay = 3 hops.

	user movements	avg. handovers /timestamp	movements /handover
train (48 users)	240	6.9	3.5
highway (50 users)	($\pm$) 325	($\pm$) 13.6	($\pm$) 2.4

The total number of handovers must be seen in relation to the number of users movements (i.e. when a user changes his access gateway). The results are presented in Table 1, and are given for $\alpha/\beta = 1$ and delay = 3 hops. For the highway scenario, only averages are shown, due to the random nature of the scenario. In the first column, the total number of user movements is given, the second column gives the (average) number of handovers per timestamp during the simulation. The third column shows the number of user movements before a handover becomes necessary. It might be no surprise that this number averages around 3. The presented results are for a 3 hops delay constraint, so in most cases an application server switch might be necessary after 3 movements . In the train scenario, this number is slightly higher, due to the fact that all users arrive at the central station. Their next hop is the office, which is mostly more than 3 hops away from the central station. An additional handover is necessary for their final movement. In the highway scenario, this is less the case. The server to which the user is connected when he leaves the highway (which is nearby his office) is probably also within 3 hops from the office, so no final handover is necessary.

We can conclude that in our model, a larger delay (such that more server locations are reachable from an AGW within the delay constraint) does not always result in a decrease of the number of handovers.

4 Application server handover mechanism

From the results given in Table 1, we can conclude that a lot of handovers are necessary in a thin-client network. Users should not notice their running applications migrating to another server. A handover mechanism with minimal duration is needed.

4.1 Migration of thin-client applications

A process migration should leave no residual state at the original server and the application must run independently on the new server. An overview of existing migration architectures was presented in section 2. Several other problems arise when migrating a thin-client application. A monitoring platform should be deployed for the user-server connections. Triggers are fired when a connection deteriorates to start the selection of a better application server and resource reservations must be made. The process is checkpointed and the process image is transmitted to a new server, where the application is restarted. In this paper, we assume the presence of a monitoring platform, decision algorithms for the choice of a better application server for the client, a process migration architecture and a resource reservation protocol. These problems are not inherent to the thin-client architecture.

A thin-client and his server communicate over a permanent connection via a remote desktop protocol. User events are sent to the server, which renders and sends back the appropriate graphical output. In order to guarantee state consistency, it is important that no user events are lost. When a process is checkpointed, a notice should be sent to the user with the IP address of the new application server. The application updates its connection status and future user events are sent to the correct application server. Several examples of connection status update mechanisms were given in section 2.

User events can get lost in two ways: events sent just after the process is checkpointed, but before the notice from the (original) application server is received; and events, sent after notice reception, that arrive at the new application server before the process image is restarted there. It should be noted that most thin-client protocols run above TCP, so delivery of the user events is guaranteed. We believe that is better not to rely on the TCP retransmit. Latency and responsiveness are of great importance in a thin-client environment, so waiting on a time-out of the TCP timers introduces unnecessary delays. Furthermore, time-outs trigger the congestion control mechanism of TCP, which is a wrong reaction since the data loss is not the result of a congestion problem inside the network.

4.2 Handover models

Figure 4 presents 3 possible models for the handover. The application migrates from server 1 to server 2. For all situations, we calculate the down-time of the application or, otherwise stated, the timeframe during which the client does not receive any graphical updates from its application server. It is assumed that the IP address of the next application server is known at $t = 0$. The user is continuously sending events to the server. Other assumptions are the immediate processing of incoming user events and immediate availability of the graphical results to be sent back. Table 2 gives an overview of the used parameters. We assume a symmetric one-way path delay between user and server. Please note that $\delta_1 \geq \delta_2$, otherwise a handover would not be necessary.

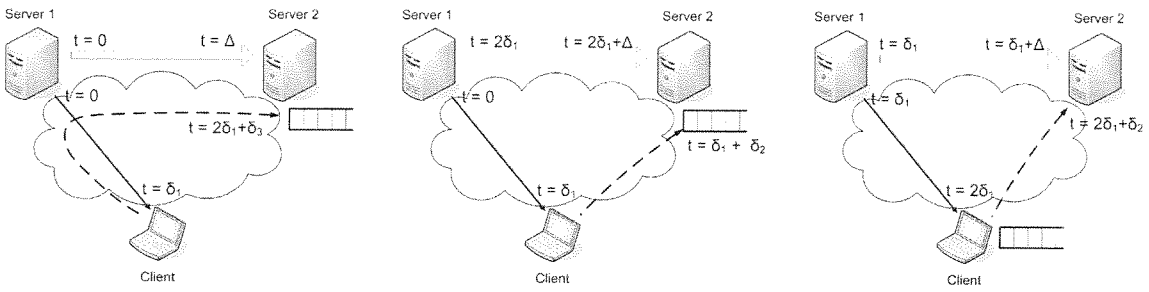
Table 2: Used parameters in the 3 situations of Figure 4. The one-way path delay is assumed symmetric.

δ_1	one-way client - server 1 path delay
δ_2	one-way client - server 2 path delay
δ_3	one-way server1 - server 2 path delay
Δ	time to checkpoint, transmit and restart application

4.2.1 Event forwarding

Figure 4(a) shows the considered configuration. At $t = 0$, the process is checkpointed and the client receives its last screen updates from server 1 at $t = \delta_1$. At the moment of checkpointing, a notice is sent to the client to inform it of the migration. This notice contains the IP of the new application server. Events generated after notice reception are sent to server 2. All user events, generated between the moment of checkpointing ($t = 0$) and notice reception ($t = \delta_1$) are still sent to server 1, which immediately forwards them to server 2. In worst case, the last forwarded user event arrives at server 2 at $t = 2\delta_1 + \delta_3$. The application is again up and running on server 2 at $t = \Delta$. This results in a timeframe of $T = \max(2\delta_1 + \delta_3, \Delta) + \delta_2 - \delta_1$ during which the client does not receive screen updates.

In this scenario, a buffer must be provided at server 2 during the handover. The period $\tau = 2\delta_1 + \delta_3$ must be accurately measured, since only after this period all user events are forwarded by server 1. The first event directly sent to server 2 arrives at $t = \delta_1 + \delta_2$, but to guarantee state consistency, these events may not be processed before the ones that were



(a) Event forwarding (b) Delayed checkpointing (c) Client buffering

Figure 4: 3 handover models

forwarded by server 1. The measurement of τ can probably be accomplished with the well-known *ping* utility, although the accuracy of this method should be verified. Based on the measurement variation, one could introduce a safety margin ϵ and wait $\tau + \epsilon$ before processing the events directly sent to server 2. Every additional delay however deteriorates the responsiveness of the application.

4.2.2 Delayed checkpointing

Figure 4(b) shows the second scenario. At $t = 0$, a notice is sent to the client with the IP of the new application server. The checkpointing itself is delayed till $t = 2\delta_1$, the last moment on which user events can arrive on server 1. We assume that the graphical response to these events is sent back to the client just before the checkpointing. These updates arrive at the client at $t = 3\delta_1$. The migrated application is up and running on server 2 at $t = 2\delta_1 + \Delta$. The first user events arrive at server 2 at $t = \delta_1 + \delta_2$, so this server can sent back graphical updates starting at $t = \max(2\delta_1 + \Delta, \delta_1 + \delta_2)$. The client doesn't receive screen updates during an interval $T = \max(2\delta_1 + \Delta, \delta_1 + \delta_2) + \delta_2 - 3\delta_1$. Since we assumed $\delta_1 > \delta_2$, this expression can be further evaluated to $T = \Delta + \delta_2 - \delta_1$.
If $\Delta + \delta_2 < \delta_1$, the screen updates from server 2 arrive at the client before the last ones from server 1. The graphical output should be queued by server 2 for a period of $\delta_1 - \Delta - \delta_2$ before being sent to the client. In this case, an almost seamless handover between the application servers might be accomplished. However, Δ can only be estimated, since the exact duration of process migration depends on the application state itself, which is continuously changing. Another advantage of this scenario is that no user events are forwarded between the servers, which makes it easier to guarantee that no user events are lost. A drawback of this approach is that a buffer must be reserved on the server for an application that is not (yet) running on that server. Sufficiently safety mechanisms are needed to avoid state inconsistency and remove the buffer when application migration fails. Just as for the event forwarding scenario, the challenge of accurate time measurements is faced. Due to the delayed checkpointing, the total handover time may increase. This increases further when a safety margin is included in the checkpoint delay.

4.2.3 Client buffering

This scenario is shown in Figure 4(c). The client keeps a copy of every user event in a buffer queue, untill an acknowledge message (ACK) is received. These ACKs are generated by the server for every received user event and normally events stay for $2\delta_1$ in the queue. If an event is not ACKed within this timeframe, all non-ACKed events in the queue are resent.
The checkpoint is taken at $t = \delta_1$ and at the same moment a migration notice is sent to the client, probably piggybacked with an ACK or a screen update. Upon notice reception, all the following user events are added to the end of the buffer queue, but are not sent to server 2. Direct transmission is only resumed upon reception of a new ACK. The front of the queue contains the events generated between $t = 0$ and $t = 2\delta_1$. These were not yet ACKed, since the application was checkpointed before they arrived at server 1. At $t = 2\delta_1$, the first event in the buffer queue generates a time-out and is resent, this time to server 2. It arrives at the server at $t = 2\delta_1 + \delta_2$. The application is up and running on server 2 at $t = \delta_1 + \Delta$. If $\delta_1 + \delta_2 > \Delta$, the retransmitted event is received, immediately processed and ACKed by server 2. The first screen update arrives at the client at $t = 2\delta_1 + 2\delta_2$, which means that no screen refreshments were received during a timeframe of $T = 2\delta_2$. If $\delta_1 + \delta_2 < \Delta$, $\lceil(\Delta - \delta_1 - \delta_2)/2\delta_1)\rceil$ retransmits of the user events are necessary, where $\lceil X \rceil$ denotes the smallest integer larger than or equal to X. The first screen update arrives at $t = 2\delta_1 + (\lceil(\Delta - \delta_1 - \delta_2)/2\delta_1\rceil - 1)\,2\delta_1 + 2\delta_2$. Retransmits occur every $2\delta_1$ and the last retransmit is successfull. This means that no screen updates are received during a timeframe of $T = (\lceil(\Delta - \delta_1 - \delta_2)/2\delta_1\rceil - 1)\,2\delta_1 + 2\delta_2$.

The drawback of this scenario is that an acknowledgement, time-out and retransmit mechanism must be added to the thin-client protocol. In section 4.1, we argued not to rely on the TCP time-out mechanism, because the wrong actions are taken by the protocol. However implementing this function at the application layer might be more efficient, questions can be raised by the presence of the same function at two layers and the required modifications of existing thin-client protocols.

Table 3: Summary of the results for the 3 scenarios of Figure 4.

scenario	*application downtime*
event forwarding	$\max(2\delta_1 + \delta_3, \Delta) + \delta_2 - \delta_1$
delayed checkpointing	$\Delta + \delta_2 - \delta_1$
client buffering	
$\delta_1 + \delta_2 > \Delta$	$2\delta_2$
$\delta_1 + \delta_2 < \Delta$	$(\lceil(\Delta - \delta_1 - \delta_2)/2\delta_1\rceil - 1)\,2\delta_1 + 2\delta_2$

5 Conclusion and future work

Thin-client user responsiveness can only be guaranteed if the application is executed on a nearby application server. Applications should make a parallel movement with the mobile users and migrate from one server to another. In this paper, the number of handovers resulting from our model, was calculated for a city network. Results were presented for both a highway and a train scenario. The conclusion was drawn that in our model a greater allowable delay does not necessarily lead to a decrease of the number of servers. The precise location of the different application servers is a determing factor. In order to minimize the number of handovers, the model should be extended with an extra term that introduces a penalty for each handover.
We only considered the hop count as a decisive parameter to initiate a handover between two application servers. Of course, other parameters also influence the experienced user responsiveness. The introduction of these parameters can be a second extension of the presented model.

Ideally, the user does not notice the application handover. Three possible mechanisms were presented and the time was calculated during which the client doesn't receive screen updates. These differ in terms of buffer locations, forwarding of user events and timing of the application checkpointing. An accurate time measurement tool is necessary for the proper functioning of the presented mechanisms. Our future work will be mainly oriented towards the development of a high performance decision algorithm to choose an appropriate application server and a protocol to reserve resources on the application server previous to the migration.

References

[1] Litzkow M. and Livny M. "Supporting Checkpointing and Process Migration Outside the UNIX Kernel". In *Proceedings of the Winter 1992 USENIX Conference*, San Francisco, January 1992.

[2] Barak A. and Wheeler R. "MOSIX: An Integrated Multiprocessor UNIX". In *Proceedings of the USENIX Winter 1989 Technical Conference*, San Diego, February 1989.

[3] Osman S., Subhraveti D., Su G., and Nieh J. "The Design and Implementation of Zap: A System for Migrating Computing Environment". In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, December 2002.

[4] Su G. *"MOVE: Mobility with Persistent Network Connections"*. PhD thesis, Department of Computer Science, Columbia University, October 2004.

[5] Kuntz B. and Rajan k. "MIGSOCK: Migratable TCP socket in Linux", February 2002.

[6] Lee K. "Migsock vs Zap", Carnegie Mellon University, Pittsburgh.

[7] De Greve F., Van Quickenborne F., De Turck F., Moerman I., and Demeester P. "Aggregation Network Design for Offering Multimedia Services to Fast Moving Users". In *Quality of Service in multiservice IP networks, Proceedings lecture notes in computer science*. Springer-Verlag Berlin, 2005.

[8] Deboosere L., Simoens P., Dewinter D., and De Turck F. "Dimensioning a Wide-Area Thin-Client Computing Network Supporting Mobile Users". In *Accepted for publication at the International Conference on Networking and Services*, Silicon Valley, July 2006.

[9] Lai A. and Nieh J. "Limits of Wide Area Thin Client Computing". In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*, Marina del Rey, 2002.

[10] Baratto R., Kim L., and Nieh J. "THINC: A Virtual Display Architecture for Thin-Client Computing". In *the Twentieth ACM Symposium on Operating Systems Principles (SOSP 2005)*, Brighton, United Kingdom, October 2005.

[11] Baratto R., Potter S., Su G., and Nieh J. "MobiDesk: Mobile Virtual Desktop Computing". In *Proceedings of the 10th ACM International Conference on Mobile Computing and Networking (MobiCom 2004)*, Philadelphia, September 2004.

PSC'06

The 2006 International Conference on Pervasive Systems &Computing

Session: MOBILE ACCESS + MOBILE DEVICES + MOBILITY MANAGEMENT

Session: SENSOR NETWORKS + ADD HOC NETWORKS

Session: ALGORITHMS AND TOOLS

Session: CONTEXT-AWARE AND RELATED ISSUES

The 2006 World Congress in Computer Science, Computer Engineering, and Applied Computing

Monte Carlo Resort, Las Vegas, Nevada, USA
June 26–29, 2006

Conferences:

The 2006 International Conference on Bioinformatics & Computational Biology

The 2006 International Conference on Computer Design & International Conference on Computing in Nanotechnology

The 2006 International Conference on Computer Graphics & Virtual Reality

The 2006 International Conference on Communications in Computing

The 2006 International Conference on Scientific Computing

The 2006 International Conference on Data Mining

The 2006 International Conference on e-Learning, e-Business, Enterprise Information Systems, e-Government, & Outsourcing

The 2006 International Conference on Engineering of Reconfigurable Systems & Algorithms

The 2006 International Conference on Embedded Systems & Applications

The 2006 International Conference on Foundations of Computer Science

The 2006 International Conference on Frontiers in Education: Computer Science & Computer Engineering

The 2006 International Conference on Grid Computing & Applications

The 2006 International Conference on Artificial Intelligence

The 2006 International Conference on Internet Computing & International Conference on Computer Games Development

The 2006 International Conference on Wireless Networks

The 2006 International Conference on Information & Knowledge Engineering

The 2006 International Conference on Image Processing, Computer Vision, & Pattern Recognition

The 2006 International Conference on Machine Learning: Models, Technologies & Applications

The 2006 International Conference on Modeling, Simulation & Visualization Methods

The 2006 International Conference on Parallel & Distributed Processing Techniques & Applications & International Conference on Real-Time Computing Systems & Applications

The 2006 International Conference on Pervasive Systems & Computing

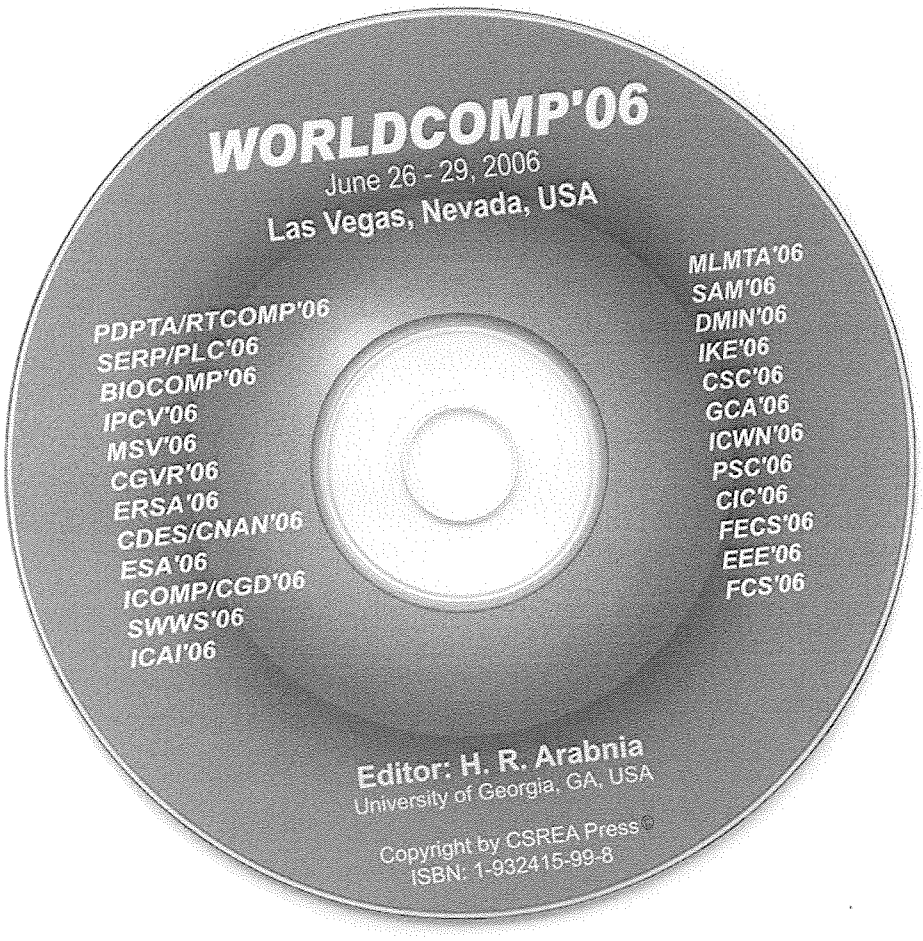
The 2006 International Conference on Security & Management

The 2006 International Conference on Software Engineering Research & Practice & International Conference on Programming Languages and Compilers

The 2006 International Conference on Semantic Web & Web Services

Editor H.R. Arabnia
University of Georgia, GA, USA

ISBN: 1-932415-99-8


WORLDCOMP'06
June 26 - 29, 2006
Las Vegas, Nevada, USA
PDPTA/RTCOMP'06
SERP/PLC'06
BIOCOMP'06
IPCV'06
MSV'06
CGVR'06
ERSA'06
CDES/CNAN'06
ESA'06
ICOMP/CGD'06
SWWS'06
ICAI'06
MLMTA'06
SAM'06
DMIN'06
IKE'06
CSC'06
GCA'06
ICWN'06
PSC'06
CIC'06
FECS'06
EEE'06
FCS'06
Editor: H. R. Arabnia
University of Georgia, GA, USA
Copyright by CSREA Press©
ISBN: 1-932415-99-8