

Design and Performance Modeling of an Efficient Remote Collaboration System

Chun-Yi Tsai* and Wei-Lung Huang

**Department of Computer Science and Information Engineering,
National Taitung University, Taitung, Taiwan, R.O.C
cytsai@nttu.edu.tw*

*Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan, R.O.C
d97012@csie.ntu.edu.tw*

Abstract

Remote Desktop Protocol (RDP) provides remote login and desktop control capabilities that enable a client to completely control and access a remote server. The protocol is implemented by Microsoft Corporation based on ITU-T T.120 family protocols. The major advantage distinguishing the RDP from other remote desktop schemes, such as the frame-buffer approach, is that the protocol is based on preferably sending graphic device interface (GDI) information from a server, instead of full bitmap images. This paper proposes a remote collaboration system based, designed and implemented on RDP architecture. The proposed system consisting of a cross control scheme with bus re-encryption architecture provides the full capabilities of all remote collaborations, such as remote control, application access and sharing among all RDP clients. In addition, it provides the RDP packets recording and replaying scheme to for replaying all of the detailed tracks through which a multiparty conference traveled. The performance modeling and evaluation show the practicality of the system and that it offers a scalable model for modern cloud services.

1. Introduction

As the rapid improvement of computer performance and networking technology, networking service providers are able to provide customers more computing powers compared with that offered in past days. The concept of cloud computing for applications and services was proposed in such a mature proposition. Despite many service types that a cloud can provide, its overall resources are always finite. This implies that how to fully exploit these finite resources to achieve equilibrium on the dynamic resource allocation is a crucial area of interest for cloud service providers. Therefore, technologies of resource virtualization along with central management mechanisms[3,4] to facilitate resource utilization become researched issues. On the other hand, users are likely to exhibit a great desire to use services if cloud service providers can provide individual services as well as interactive and sharing services among users, such as social networking applications incorporating a remote desktop interface[1,2,3]. Based on demands of both service providers and users, the proposed system provides effective interactivity, concurrent sharing, and management for multi-party users. To achieve this, the Remote Desktop Protocol (RDP) [6,7] that evolved from ITU-T T.120[10,11] family protocols specified for networking conference[23,24,25] was adopted to design the proposed system. RDP allows a client to login to a server by using a user account on this server via a remote session, and then control the server session remotely as well as the behavior of sitting in front of the server machine. Disregarding the authentication and security parts, the basic operations of RDP protocol can be described as follows. First, the user input events from

the mouse and keyboard on a client are packaged into RDP input packets and sent to the server. Second, once the input event packet is received, the server unpacks it and interprets the keyboard and mouse event by keyboard and mouse driver at the server end. The server then uses the local video driver to construct the rendering information into RDP rendering packets, and returns them to the client. At the client end, once the RDP packets of the rendering data are received, the client interprets the RDP packets into corresponding GDI function calls. Finally, these GDI function calls are applied at the client end to display the rendering data on screen. Because the data transmitted between the client and server is primarily GDI information encapsulated in RDP packets, the total amount of data transmitted by the RDP can be reduced compared with the remote frame buffer[5] scheme. The remainder of this paper is organized as follows: section 2 presents RDP and introduces the system basics, section 3 discusses the design details of the proposed system, section 4 shows demonstration, section 5 elaborates the performance modeling and evaluation, and the conclusion is given in section 6.

2. System Basics and Related Work

2.1. Session Key Negotiation of RDP

As a ciphering-based session, the first step of the RDP initiating communication between an RDP client and the RDP server is the session key negotiation, illustrated in Figure 1.

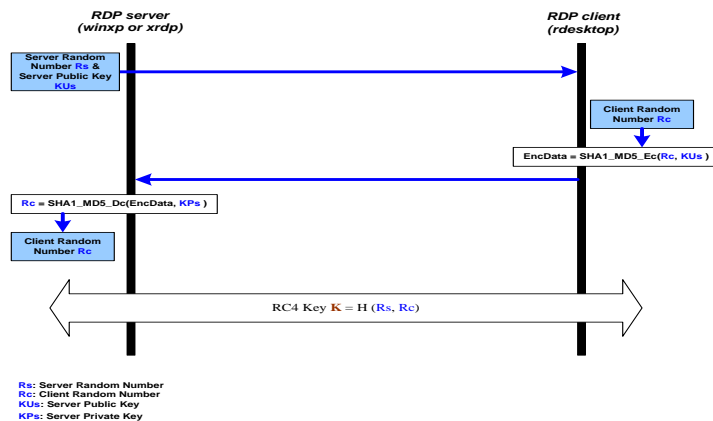


Figure 1. Session Key Negotiation of RDP

The server first sends an unencrypted server random number R_s and a server public key KUs to the client. When the client receives the R_s and KUs , it randomly generates a client random number R_c , then encrypts R_c with KUs by SHA1 and MD5 encryption and sends it to the server. When the encrypted data is received by the server, the server decrypts the data with its own server private key KPs and thus obtains the R_c . Since both the server and client get R_s and R_c , a RC4 key K can be computed on both sides by applying a hashing function H with parameters R_s and R_c . The key K is used for RC4 encryption and decryption in the subsequent communication of this RDP session.

2.2. RDP Gateway Architecture

The basic RDP gateway for a multi-party conference is illustrated in Figure 2. Each RDP client does not directly connect to an RDP server but connects to an RDP gateway. The RDP gateway then relays the client connection setup request to the RDP server and applies the man-in-the-middle technique to obtain both the server and client random numbers during the RC4 session key negotiation phase.

Regarding the RDP server aspect, the RDP gateway acts like an RDP client. By contrast, from the client's point of view, the RDP gateway servers as an RDP server.

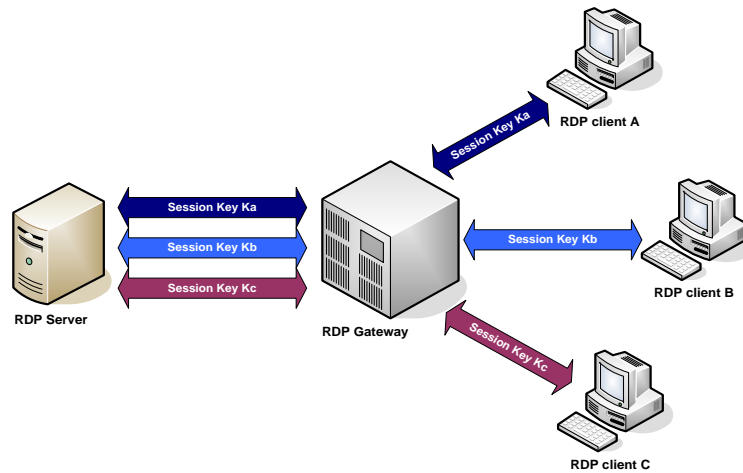


Figure 2. RDP Gateway Architecture

2.3. Negotiation via RDP Gateway

Figure 3 shows the complete process of a session key negotiation of the RDP gateway architecture. The client runs rdesktop[8] on Linux developed by Chapman who contributed another open source work of RDP server on Linux, xrdp[9]. The RDP gateway can be divided into two parts: the faked client end and faked server end. The detailed process of the session key negotiation is as follows. When the faked client end receives the actual server random number R_s and a server public key K_U on the initial step of negotiation, the faked server end sends R_s and a faked server public key K_{Uf} to the client, and generates its private key K_{Psf} at the same time. The client then generates a client random number R_c , and encrypts it with K_{Uf} by SHA1[13,15] and MD5[14,15] encryption, and returns it to the faked server end. Once the encoded data is received by the faked server end, the R_c can be decrypted with K_{Psf} . The faked client end then encrypts the R_c with K_U and sends it to the real RDP server. Finally, all of the real server end, faked server end, and faked client end get R_s and R_c , and can compute the RC4[12,15] session key K by a hashing function H with parameters R_s and R_c . Thus, the RDP gateway can successfully decrypts the incoming encrypted RDP packets transmitted between the real RDP server and RDP client with session key K by RC4 decryption, and so does the encryption. Based on the session key negotiation and the indirect relay connection architecture offered by the RDP gateway, the architecture can be further extended to develop a multi-party collaboration system, which is described in the following section.

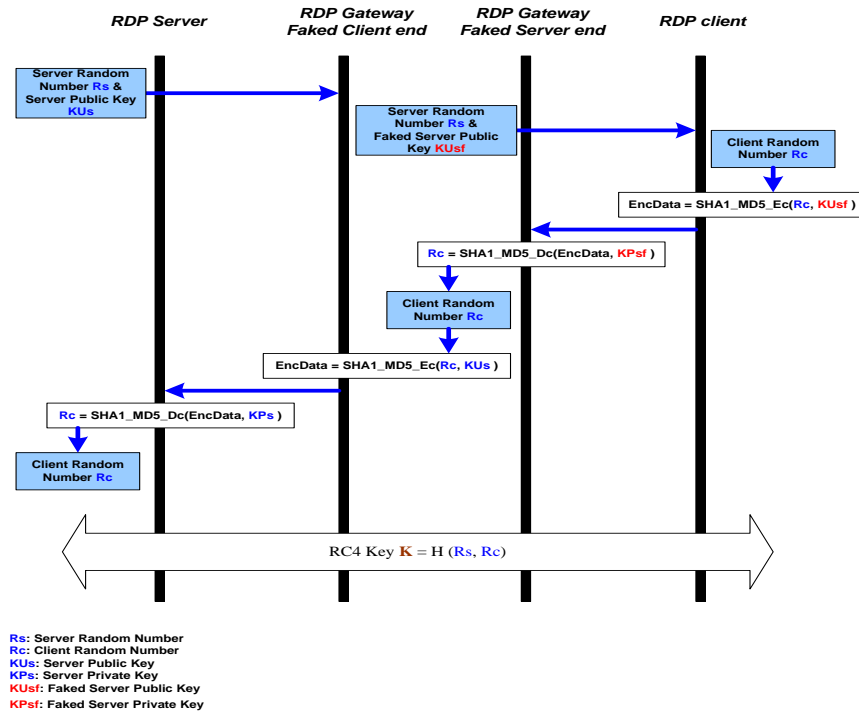


Figure 3. Session Key Negotiation via RDP Gateway

3. Design of the Proposed System

3.1. Multi-Party Cross Control Scheme

The proposed system mainly relies on the multi-party cross control scheme and re-encryption architecture design. By applying the cross control scheme, one RDP client can perform a “cross control” over another remote desktop of an RDP client and share a live desktop of this controlled client to all other clients via the RDP gateway. However, since the RC4 encryption is used for transmitting data of a connection between the client and server, the calling counts of the RC4 encryption and decryption on both ends must always keep equal in anytime during a normal live session. Otherwise, a derived problem of mismatched calling counts of encryption and decryption between the sending end of the controlled client and receiving end of the corresponding faked server on the gateway occurs when the cross control is activated. The bus re-encryption architecture was proposed to solve this problem. Figure 4 illustrates a scenario in which RDP client A attempts to control the desktop of another RDP client B, and shares the results with all other clients.

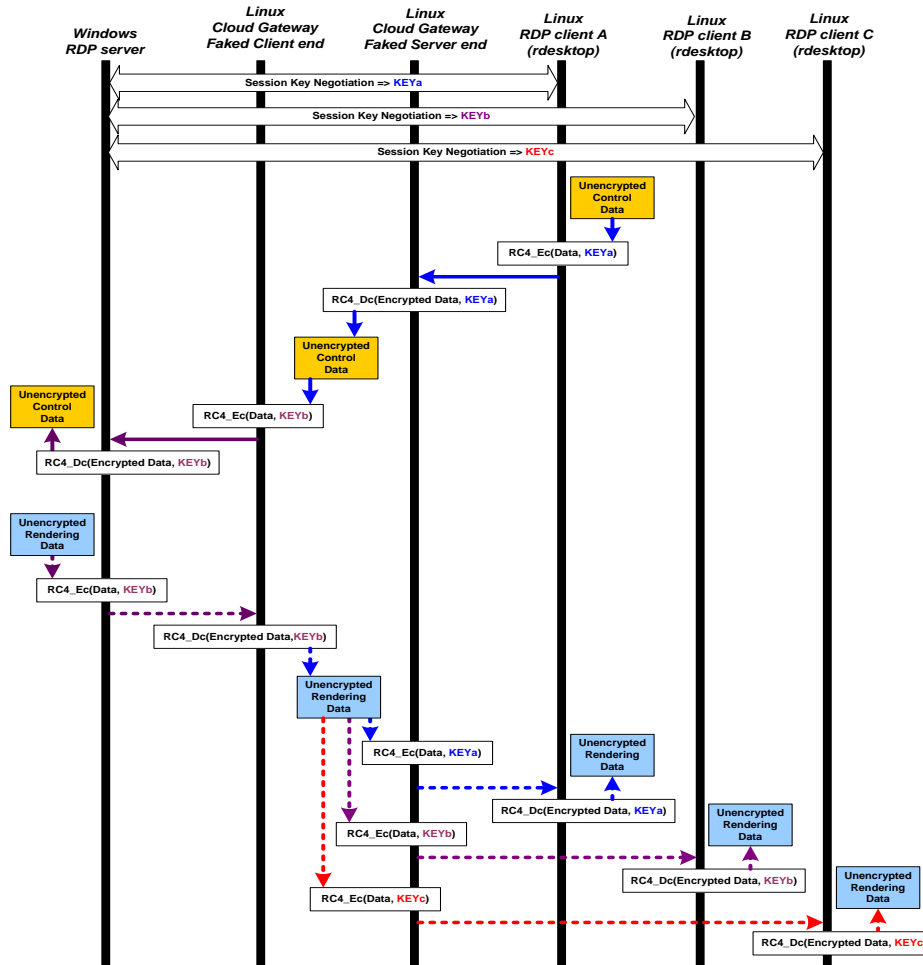


Figure 4. Multi-Party Cross-Control Scheme

First, the session key negotiations of client A, client B and client C are processed, and they obtain their negotiated session key, KEYa, KEYb, and KEYc respectively. Then client A attempts to control the desktop of client B by sending the RDP packets of control messages encrypted with KEYa and thus the control message is then decrypted by the faked server end of the RDP gateway. To serve as sending control message from client B, the control message must be re-encrypted with KEYb on the faked client end before being sent to the real RDP server. When the RDP packets of the rendering data returned by the real RDP server are received and decrypted on the faked client end, the faked server end finally re-encrypts the data with KEYa, KEYb, and KEYc, and returns them to client A, client B, to client C, respectively. Eventually, the live desktop of client B can be controlled on client A and also displayed on the screens of all clients concurrently.

3.2. Bus Re-encryption Architecture

The bus re-encryption architecture is illustrated in Figure 5. The RDP gateway provides four access points for each connection such that one point maps to the receiving end of the client, another point maps to the sending end of the client, and the other two share similar mappings to the server. A crucial concern is to maintain the calling counts of the RC4 encryption and decryption to be explicitly equal at each pair of access points. This can be achieved when the unencrypted data is ready to be re-encrypted; the data is duplicated and re-encrypted by the access point that is on the path to its destination.

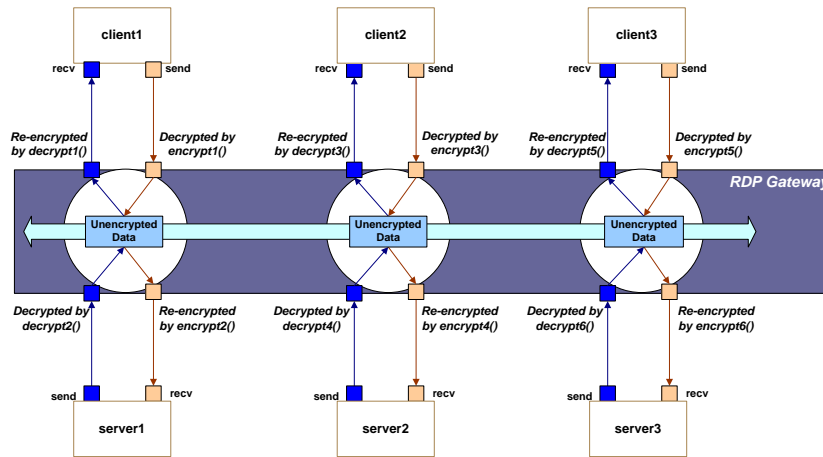


Figure 5. Bus Re-encryption Architecture

The bus architecture differ from the original usage of the normal RC4 encryption and decryption in that it maintains the same calling count of the access point as that of its mapping end, instead of maintaining the calling count of the client end the same as that of the server end. By applying the bus re-encryption architecture, the system becomes robust and does not crash due to the discordant calling counts. The principal process of the controlled client when applying the cross control scheme based on the bus re-encryption architecture is elucidated by the algorithm in Table 1. Each encrypted data received from the server of the controlled-client is first decrypted by its own decrypt function (decrypt2). Then the unencrypted data is then re-encrypted by the encryption functions (decrypt1) of all other access points and sent to all of the other clients. Subsequently, the unencrypted data is also re-encrypted by the encryption function of the access points belonging to the controlled client, and then sent to the receiving end of the controlled client. The algorithm for each thread presented in Table 1 is to perform the aforementioned process which is corresponding to each RC4 session.

Table 1. Bus Re-Encryption Algorithm

```

Algorithm Bus re-encryption
if (p->role == 1) //if the role is the controlled-client {
    decrypt(p->buffer, p->len, decrypt2, p->conn);

    for(p->i=0; p->i < TOTAL_CUR_CONN; p->i++){
        if (p->i != p->conn){
            memcpy(p->buffer2, p->buffer, p->len);

            //re-encryption
            decrypt1(p->buffer2+ rdp_conn[p->conn].server_pkt_skip,
                p->len- rdp_conn[p->conn].server_pkt_skip, p->i);

            //send to other clients
            send(client->[p->i], p->buffer2, p->len, 0);
        }
    }

    //re-encryption
    decrypt1(p->buffer+ rdp_conn[p->conn].server_pkt_skip,
        p->len- rdp_conn[p->conn].server_pkt_skip, p->conn);

    //send to the controlled-client
    send(client[p->conn], p->buffer, p->len, 0);
}
    
```

3.3. Remote Desktop Recording and Replay

Another markedly practical function of the proposed system is the provision of remote desktop recording for every moment of live multi-party conferences and off-line replays of them, which can be performed in three phases. During the first phase, a recording process first logs every RDP packet sent from an RDP server during a live RDP session. A first-time-replaying process during the second phase then renders the GDI information contained in the RDP packets logged during the first process. After this replay, a crucial process called referencing points and frames logging is saved to maintain the vital referencing information for later playback control. Thus, during the third phase, users can apply playback control functionality, including the forward, rewind and temporal stop controls when replaying the recorded conference. As shown in Figure 6, the proposed system consists of a storage system to store the unencrypted rendering data of every live RDP packet sent from the server to client during the recording process. The recorded unencrypted data is not only a static log, but can be replayed later by enabling the replaying process as illustrated at the bottom half of the figure. To accomplish this, the client renegotiates a new session key with the RDP gateway and uses it to decrypt and replay the rendering data sent from the storage system. Thus, a desktop replay of all detailed tracks through which a multiparty conference travelled can be displayed on the client screens.

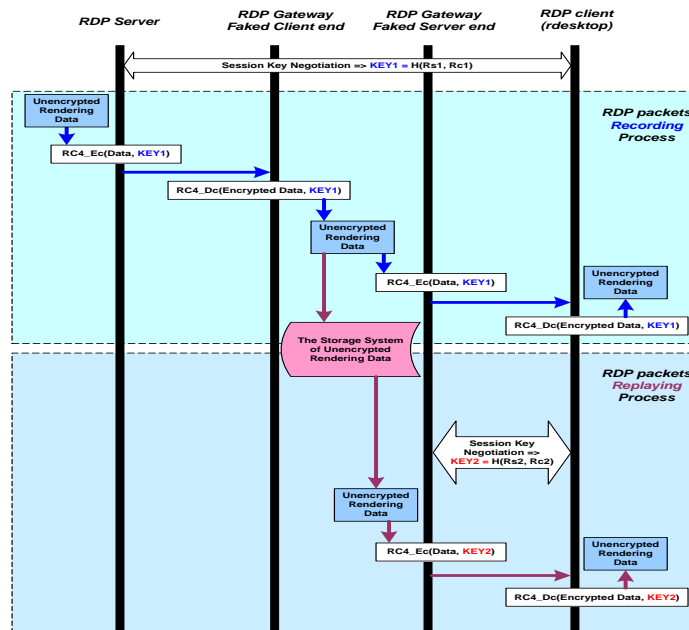


Figure 6. RDP Packets Recording and Replaying System

3.3.1. The First-Time Replay

Before the replay process begins, a new session key is newly negotiated between the RDP client and gateway. The RDP gateway then encrypts the saved unencrypted rendering data by using the new session key, encapsulates the data into RDP packets, and sends them to the client. Once the client receives RDP packets, it decrypts the packets, reads the unencrypted rendering data, renders the data, and achieves the goal of replay. When the first-time replay starts, the recording process periodically saves a complete snapshot of the working window of the client for each RDP packet received from the gateway, as shown in Table 2.

Table 2. RDP-Packets-Recording Algorithm

```
Algorithm RDP-packets-recording  
void ui_full_desktop_save (uint32 offset, int x, int y, int cx, int cy){  
    Pixmap pix;  
    XImage *image;  
    uint32 offset = 0;  
    ...  
    image = XGetImage(g_display, g_backstore, 0, 0, cx, cy, AllPlanes, ZPixmap);  
    exit_if_null(image);  
    }  
    offset *= g_bpp / 8;  
    cache_put_full_desktop (offset, cx, cy, image-> bytes_per_line, g_bpp / 8, (uint8 *) image->  
>data);  
    ...  
    XDestroyImage(image);  
    }  
void cache_put_full_desktop(uint32 offset, int cx, int cy, int scanline, int bytes_per_pixel,  
uint8 * data)  
{  
    int length = cx * cy * bytes_per_pixel;
```

3.3.2. Playback Control of Replay

The common and necessary playback control of replay consists of forward, rewind, fast forward, and fast rewind functionalities. Because the RDP protocol is not based on a frame-buffer design in which each frame can be rendered independently, the RDP client instead renders the GDI information in an accumulative fashion. For example, all of the GDI information on menus, buttons, images, and objects of a window can be contained in distinct RDP packets and rendered in the receiving order. Thus, the replay is likely to cause broken windows or other strange appearances if the playback begins from some intermediate RDP packet instead of beginning from the first packet sent from the gateway. The referencing frames saved during the first-time replay mentioned in the previous subsection were adopted to solve this problem. The referencing frames are full desktop bitmap images that are snapshot of each RDP packet. The forward or rewind playback from a specific point is performed by beginning from the nearest referencing point and reloading the corresponding saved referencing frame as presented in Table 3.

Table 3. Rdp-Packets-Reloading

```
Algorithm RDP-packets-reloading  
uint8* cache_get_full_desktop(uint32 offset, int cx, int cy, int bytes_per_pixel, unsigned int  
snpst_no){  
    int length = cx * cy * bytes_per_pixel;  
    fseek(fp_full_desktop_snapshot, length* snpst_no, SEEK_SET);  
    fread(g_full_deskcache, length, 1, fp_full_desktop_snapshot);  
    if (offset > sizeof(g_full_deskcache))  
        offset = 0;  
    if ((offset + length) <= sizeof(g_full_deskcache))  
        return &g_full_deskcache[offset];  
    error(get_full_desktop "%d:%d\n", offset, length);  
    return NULL;  
    }
```

4. Demonstration

The test-bed environment used for demonstration consisted of the following platforms and components. The gateway ran Linux kernel 2.6.38 on a Core i7 3.07 GHz desktop containing 4 GB RAM and a sever machine running Windows OS on a Core i7 3.07GHz desktop containing 4 GB RAM. For a convenient demonstration, the controller, controlled client, and other general clients were run on a Linux machine running kernel 2.6.38 and

equipped with a Celeron 330 2.5 GHz processor and 2 GB RAM. Another general thin-client was run on an iPad tablet.

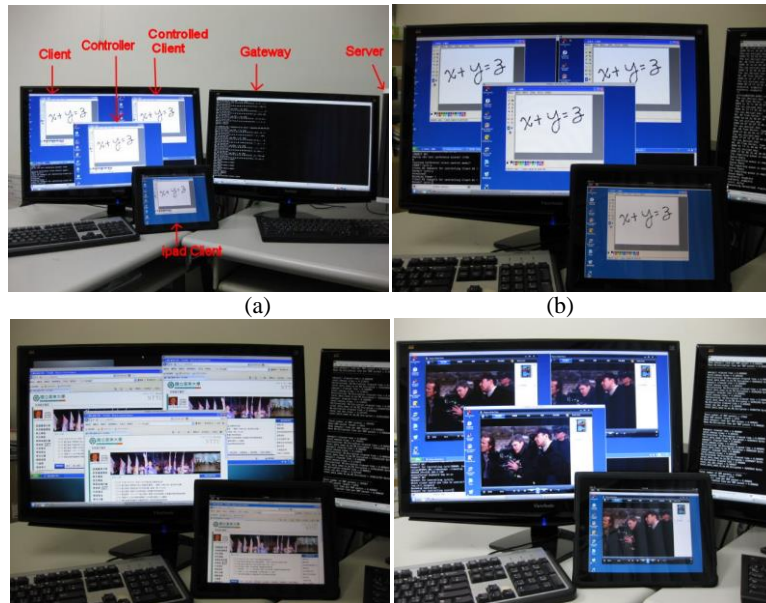


Figure 7. (a) Experimental Environment (b) Paint Applications (c) Web Browsing (d) Video Playing

Figure 7a and 7b indicate that the paint application executed satisfactorily in either a self-control/sharing mode or cross-control/sharing mode. Web applications shown in Figure 7c demonstrated favorable performance, and the video shown in Figure 7d ran considerably smoothly. Actually, all of the applications provisioned on the controlled server, to which the controlled session connected, could be completely controlled and applied by the controller, and any change in the desktop display of the controlled session was shared and delivered to all of the clients concurrently. Thus, all of the users could observe the same desktop display immediately on any of the client screens.

5. Performance Modeling and Evaluation

5.1. Queuing Model

The core system consisting of a gateway and a controlled server, can be modeled as an open queuing network [16- 22], as illustrated in Figure 8.

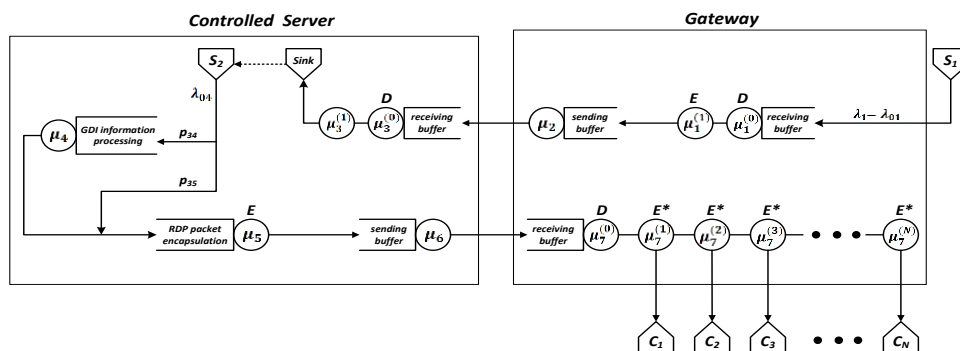


Figure 8. The Proposed Open Queuing Network For Performance Modeling

queueing network [16- 22], as illustrated in Figure 8.

The network consists of seven nodes, and each node is referred to as an $M/M/1/FCFS$ queue for processing RDP packets. The parameters and symbols are denoted in Figure 8, and those that appear in the subsequent analysis are defined as follows:

- S_1 : source representing the sending end of the controlling client,
- S_2 : source representing the controlled server,
- C_j : sinks representing the receiving ends for each client ($1 \leq j \leq N$),
- D : RC4 decryption service,
- E : RC4 re-encryption service,
- E^* : memcopy() service and RC4 re-encryption service,
- λ_{0i} : arrival rate from external to the i^{th} node,
- λ_i : overall arrival rate at the i^{th} node,
- μ_i : service rate at the i^{th} node,
- $\mu_i^{(m)}$: service rate of the m^{th} server in sequence at the i^{th} node,
- p_{ij} : routing probability that a packet is delivered to the j^{th} node from the i^{th} node,
- ρ_i : traffic intensity at the i^{th} node,
- \bar{K}_i : mean queue length at the i^{th} node,
- \bar{T}_i : mean response time at the i^{th} node, and
- N : the total number of clients.

This open queueing network can be solved by applying the following steps. First, the arrival rate at each node can be obtained by

$$\begin{aligned} \lambda_3 &= \lambda_2 = \lambda_1 = \lambda_{01}, & \lambda_4 &= p_{34}\lambda_{04}, \\ \lambda_5 &= \lambda_4 + p_{35}\lambda_{04} = p_{34}\lambda_{04} + p_{35}\lambda_{04} = \lambda_{04}, & \lambda_7 &= \lambda_6 = \lambda_5 = \lambda_{04}. \end{aligned}$$

The traffic intensity at each node then becomes

$$\begin{aligned} \rho_1 &= \frac{\lambda_1}{\mu_1} = \lambda_{01} \left(\frac{1}{\mu_1^{(0)}} + \frac{1}{\mu_1^{(1)}} \right) = \frac{\lambda_{01}(\mu_1^{(0)} + \mu_1^{(1)})}{\mu_1^{(0)}\mu_1^{(1)}}, & \rho_2 &= \frac{\lambda_2}{\mu_2} = \frac{\lambda_{01}}{\mu_2}, \\ \rho_3 &= \frac{\lambda_3}{\mu_3} = \lambda_{01} \left(\frac{1}{\mu_3^{(0)}} + \frac{1}{\mu_3^{(1)}} \right) = \frac{\lambda_{01}(\mu_3^{(0)} + \mu_3^{(1)})}{\mu_3^{(0)}\mu_3^{(1)}}, & \rho_4 &= \frac{\lambda_4}{\mu_4} = \frac{p_{34}\lambda_{04}}{\mu_4}, & \rho_5 &= \frac{\lambda_5}{\mu_5} = \frac{\lambda_{04}}{\mu_5} \\ \rho_6 &= \frac{\lambda_6}{\mu_6} = \frac{\lambda_{04}}{\mu_6}, & \rho_7 &= \frac{\lambda_7}{\mu_7} = \lambda_{04} \left(\frac{1}{\mu_7^{(0)}} + \frac{1}{\mu_7^{(1)}} + \dots + \frac{1}{\mu_7^{(N)}} \right) = \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}. \end{aligned}$$

It follows that the mean queue length of each node can be computed by

$$\begin{aligned} \bar{K}_1 &= \frac{\rho_1}{1-\rho_1} = \frac{\lambda_{01}(\mu_1^{(0)} + \mu_1^{(1)})}{\mu_1^{(0)}\mu_1^{(1)} - \lambda_{01}(\mu_1^{(0)} + \mu_1^{(1)})}, & \bar{K}_2 &= \frac{\rho_2}{1-\rho_2} = \frac{\lambda_{01}}{\mu_2 - \lambda_{01}} \\ \bar{K}_3 &= \frac{\rho_3}{1-\rho_3} = \frac{\lambda_{01}(\mu_3^{(0)} + \mu_3^{(1)})}{\mu_3^{(0)}\mu_3^{(1)} - \lambda_{01}(\mu_3^{(0)} + \mu_3^{(1)})}, & \bar{K}_4 &= \frac{\rho_4}{1-\rho_4} = \frac{p_{34}\lambda_{04}}{\mu_4 - p_{34}\lambda_{04}} \\ \bar{K}_5 &= \frac{\rho_5}{1-\rho_5} = \frac{\lambda_{04}}{\mu_5 - \lambda_{04}}, & \bar{K}_6 &= \frac{\rho_6}{1-\rho_6} = \frac{\lambda_{04}}{\mu_6 - \lambda_{04}}, & \bar{K}_7 &= \frac{\rho_7}{1-\rho_7} = \frac{\lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}}{1 - \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}}. \end{aligned}$$

Thus, by Little's Theorem, this yields the mean response time at each node as follows:

$$\begin{aligned} \bar{T}_1 &= \frac{\bar{K}_1}{\lambda_1} = \frac{\mu_1^{(0)} + \mu_1^{(1)}}{\mu_1^{(0)}\mu_1^{(1)} - \lambda_{01}(\mu_1^{(0)} + \mu_1^{(1)})}, & \bar{T}_2 &= \frac{\bar{K}_2}{\lambda_2} = \frac{1}{\mu_2 - \lambda_{01}}, \\ \bar{T}_3 &= \frac{\bar{K}_3}{\lambda_3} = \frac{\mu_3^{(0)} + \mu_3^{(1)}}{\mu_3^{(0)}\mu_3^{(1)} - \lambda_{01}(\mu_3^{(0)} + \mu_3^{(1)})}, & \bar{T}_4 &= \frac{\bar{K}_4}{\lambda_4} = \frac{1}{\mu_4 - p_{34}\lambda_{04}}, & \bar{T}_5 &= \frac{\bar{K}_5}{\lambda_5} = \frac{1}{\mu_5 - \lambda_{04}}, \\ \bar{T}_6 &= \frac{\bar{K}_6}{\lambda_6} = \frac{1}{\mu_6 - \lambda_{04}}. \end{aligned}$$

The mean response time of a packet in node 7 can then be calculated for the N th client, $\overline{T_7^{CN}}$, i.e.,

$$\overline{T_7^{CN}} = \overline{T_7} = \frac{\overline{K_7}}{\lambda_7} = \frac{\sum_{i=0}^N (\mu_7^{(i)})^{-1}}{1 - \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}}.$$

The evaluation of mean response time in node 7 for those clients, excepting for the N th client can be calculated in two parts. The first part is the actual service time denoted by $\overline{T_7^{Cj^S}}$ that

$$\overline{T_7^{Cj^S}} = \overline{T_7^{CN}} - \sum_{i=j+1}^N (\mu_7^{(i)})^{-1}, 1 \leq j \leq N - 1.$$

The next packet waiting in queue is not retrieved until the current served packet passes through the encryption chain. Thus, the mean response time for such clients in node 7 is

$$\overline{T_7^{Cj}} = \overline{T_7^{Cj^S}} + \overline{T_7^{Cj^R}} = \frac{\sum_{i=0}^N (\mu_7^{(i)})^{-1}}{1 - \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}}, 1 \leq j \leq N - 1,$$

which yields the identical result to that of $\overline{T_7^{CN}}$.

Therefore, the overall mean response time for each client is

$$\overline{T_{Cj}} = [\sum_{k=1}^6 \overline{T_k}] + \overline{T_7^{Cj}} = [\sum_{k=1}^6 \overline{T_k}] + \frac{\sum_{i=0}^N (\mu_7^{(i)})^{-1}}{1 - \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}}, 1 \leq j \leq N. \quad (1)$$

5.2. Queuing Model

Measuring from the real system in a case of real-time video streaming, all arguments required in aforementioned equations can be obtained by well-designed instrumentation codes. The detailed arguments are listed as follows:

$$\text{mean RC4 decoding time: } \frac{1}{\mu_1^{(0)}} = \frac{1}{\mu_3^{(0)}} = \frac{1}{\mu_7^{(0)}} = 3.8 \times 10^{-5} (s/pkt),$$

$$\text{mean RC4 encoding time: } \frac{1}{\mu_1^{(1)}} = \frac{1}{\mu_5} = 3 \times 10^{-5} (s/pkt),$$

$$\text{mean memcpy() time + mean RC4 encoding time: } \frac{1}{\mu_7^{(1)}} = \frac{1}{\mu_7^{(2)}} = \dots = \frac{1}{\mu_7^{(N)}} = 3.1 \times 10^{-5} (s/pkt),$$

$$\text{mean transmission time measured on Gigabit Ethernet: } \frac{1}{\mu_2} = \frac{1}{\mu_6} = 3.6 \times 10^{-4} (s/pkt),$$

$$\text{mean service time for management packets: } \frac{1}{\mu_3^{(1)}} = 10^{-6} (s/pkt),$$

$$\text{mean service time for GDI information processing: } \frac{1}{\mu_4} = 10^{-6} (s/pkt),$$

$$\text{mean arrival rate of S1 (packets issued by the controller): } \lambda_{01} = 10 (pkt/s),$$

$$\text{mean arrival rate of S2 (packets issued by the controlled server): } \lambda_{04} = 855.74 (pkt/s),$$

$$\text{mean RDP packet length: } 6778 (byte/pkt),$$

$$\text{total elapsed time of the video: } 330 (s), \text{ and}$$

$$\text{routing probabilities } p_{34} = 0.15 \text{ and } p_{35} = 0.85.$$

By applying these arguments into the model, the overall mean response time for each client can be calculated as the following.

$$\left[\sum_{k=1}^6 \overline{T_k} \right] = \frac{\mu_1^{(0)} + \mu_1^{(1)}}{\mu_1^{(0)} \mu_1^{(1)} - \lambda_{01} (\mu_1^{(0)} + \mu_1^{(1)})} + \frac{1}{\mu_2 - \lambda_{01}} + \frac{\mu_3^{(0)} + \mu_3^{(1)}}{\mu_3^{(0)} \mu_3^{(1)} - \lambda_{01} (\mu_3^{(0)} + \mu_3^{(1)})} + \frac{1}{\mu_4 - p_{34} \lambda_{04}} + \frac{1}{\mu_5 - \lambda_{04}} + \frac{1}{\mu_6 - \lambda_{04}} = 0.0010197$$

Hence,

$$\begin{aligned} \overline{T_{C_j}} &= \left[\sum_{k=1}^6 \overline{T_k} \right] + \frac{\sum_{i=0}^N (\mu_7^{(i)})^{-1}}{1 - \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}} = 0.0010197 + \frac{(\mu_7^{(0)})^{-1} + N * (\mu_7^{(1)})^{-1}}{1 - \lambda_{04} ((\mu_7^{(0)})^{-1} + N * (\mu_7^{(1)})^{-1})} \\ &= 0.0010197 + \frac{3.8 \times 10^{-5} + N * 3.1 \times 10^{-5}}{1 - 855.74 * (3.8 \times 10^{-5} + N * 3.1 \times 10^{-5})}, \quad 1 \leq j \leq N. \end{aligned}$$

Since system facilities are fixed, by investigating the results, it shows that there are two arguments that crucially impact on the overall system performance, the total number of clients N and the inter-arrival rate of data traffic generated from server, λ_{04}

5.2.1. Bound for N

For stability, the intensity $\rho_7 = \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1}$ must be less than 1 such that $\lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1} < 1$. Since all $\mu_7^{(i)}$ are identical for $1 \leq i \leq N$, we have $\lambda_{04} < \frac{1}{\sum_{i=0}^N (\mu_7^{(i)})^{-1}} < \frac{1}{(\mu_7^{(0)})^{-1} + N(\mu_7^{(1)})^{-1}}$, and thus, $N < \mu_7^{(1)} (\lambda_{04}^{-1} - (\mu_7^{(0)})^{-1})$. (2)

Given $\lambda_{04} = 855.74$, $(\mu_7^{(1)})^{-1} = 3.1 \times 10^{-5}$, and $(\mu_7^{(0)})^{-1} = 3.8 \times 10^{-5}$, the maximum number of clients to which the system can offer normal services in this case is $N < \mu_7^{(1)} (\lambda_{04}^{-1} - (\mu_7^{(0)})^{-1}) = 36.47$.

This shows that under the condition given in the aforementioned case, the bound for total number of clients. Otherwise, the dividend part of $\overline{T_7^{CN}}$ yields a negative value, and thus, eventually leads to an unstable state. Therefore,

$$\rho_7 = \lambda_{04} \sum_{i=0}^N (\mu_7^{(i)})^{-1} = 855.74 \times (3.8 \times 10^{-5} + 36 \times 3.1 \times 10^{-5}) = 0.9875,$$

and the overall mean response time for each client is

$$\overline{T_{C_j}} = 0.0010197 + \frac{3.8 \times 10^{-5} + 36 \times 3.1 \times 10^{-5}}{1 - 855.74 \times (3.8 \times 10^{-5} + 36 \times 3.1 \times 10^{-5})} = 0.093517 \text{ sec}, \quad 1 \leq j \leq N$$

5.2.2. Overall Mean Response Time

The analysis reveals that the intensity ρ_7 is decided by both λ_{04} and N , and thus determines the performance evaluated in eq.(1). The results are shown in Figure 9. As the number of clients increases, the overall mean response time per packet for each client evidently increases due to the fact that any client must wait for its next packet after the current packet passes through the entire re-encryption chain with degree N . In addition, the higher arrival rate the required service type needs, the fewer the tolerant vacancy of N is. Thus, the overall mean response time has a rapid growth in the case of bound for N under circumstances of a variety of arrival rates.

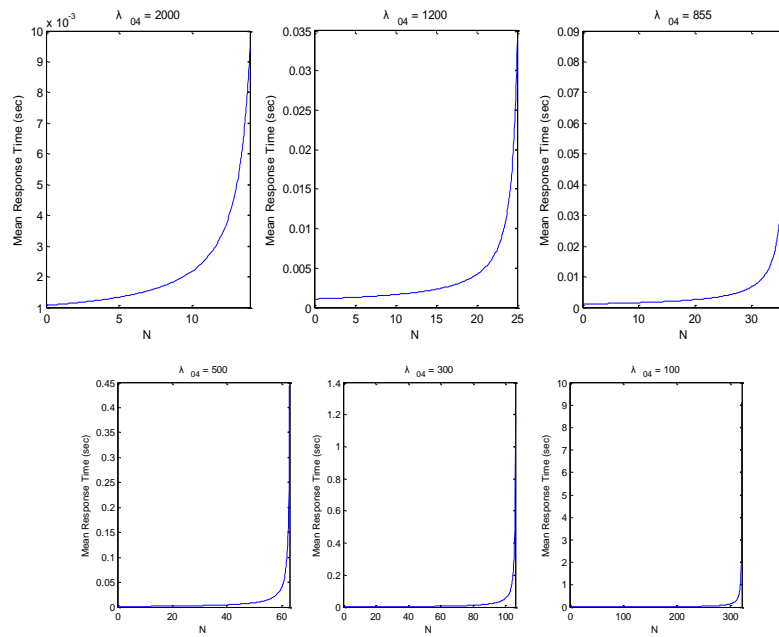


Figure 9. Overall Mean Response Time Under A Variety Of Workload Conditions

5.2.3. Marginal Probability of the Bottleneck Node

The queue capacity required for quality guarantee of a service can be reflected by the marginal probabilities of the bottleneck, node 7, in a steady state at the bound of N according to eq. (2) in various cases. As shown in Figure 10, it shows clearly that when the maximum number of clients is well-bounded to adapt to various traffic loads generated by distinct arrival rates, the marginal probability in a steady state can be under control by allocating sufficient queue capacity to the bottleneck node, and thus a high-quality service for each client can be guaranteed and the optimization of system utilization can be achieved.

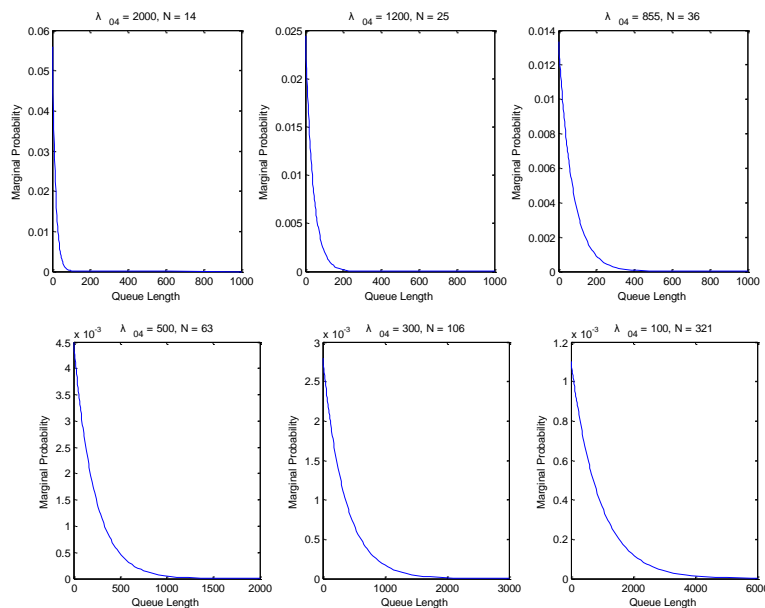


Figure 10. Marginal Probability Of The Bottleneck Node Under A Variety Of Workload Conditions

6. Conclusion

The proposed remote collaboration system is designed and implemented in accordance with the proposed cross control scheme and the bus re-encryption architecture. It is feasible to the deployment of real-time multi-party remote collaboration applications, such as networking conference, video, audio, and image accessing or sharing, remote whiteboard collaboration, etc. The major advantage of the proposed system is that all available resources on each client are able to be accessed and shared by other clients concurrently such that all resources available on the system can be fully accessed and shared. The performance modeling and evaluation measured for the system run on the testbed machine not only give a benchmark for a basic RDP collaboration system, but offer cloud providers a scalability model to deploy their devices and facilities for cloud services based on RDP. As the servers tend to be virtualized and the clients are thought to be desktops from the aspect of modern cloud computing, the proposed architecture in this work can be still applied and evaluated.

References

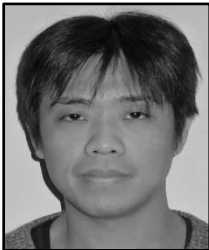
- [1] K. Beaty, A. Kochut and H. Shaikh, "(2009) Desktop to Cloud Transforming Planning", In IEEE International Symposium on Parallel & Distributed Processing, (2009) May, pp. 23-29.
- [2] Rhee J, Kochut A and Beaty K (2009) DeskBench: flexible virtual desktop benchmarking toolkit. In: The 11th IFIP/IEEE international conference on Symposium on Integrated Network Management. 1-5 June 2009.
- [3] Desktone Inc. Desktop as a Service. <http://www.desktone.com>. Accessed 20 Jan 2015.
- [4] VMWare Inc. VMWare ThinApp. <http://www.vmware.com/products/thinapp/overview.html>. Accessed 20 Jan 2015.
- [5] RealVNC Inc. The RFB Protocol. <http://www.realvnc.com/docs/rfbproto.pdf> Accessed 20 Jan 2015.
- [6] Microsoft Inc. Remote Desktop Protocol . <http://msdn.microsoft.com/en-us/library/aa383015.aspx>. Accessed 20 Jan 2015.
- [7] Microsoft Inc. Remote Desktop Protocol: Basic Connectivity and Graphics Remoting Specification. <http://msdn.microsoft.com/en-us/library/cc240445.aspx>. Accessed 20 Jan 2015.
- [8] Matthew Chapman rdesktop: A remote Desktop Protocol client. <http://www.rdesktop.org>. Accessed 20 Jan. 2015.
- [9] Mathew Chapman xrdp: An open source remote desktop protocol (rdp) server. <http://xrdp.sourceforge.net>. Accessed 20 Jan. 2015
- [10] ITU-T Recommendation T.128 Multipoint Application Sharing. Feb 1998.
- [11] ITU-T Recommendation T.125 Multipoint Communication Service Protocol Specification. Feb 1998.
- [12] IETF Draft: A Stream Cipher Encryption Algorithm Arcfour. Dec 1999.
- [13] Electronic Frontier Foundation Specifications for a Secure Hash Standard (SHS). Jan 1992.
- [14] IETF RFC 1321: The MD5 Message-Digest Algorithm. Aug 1992
- [15] The OpenSSL Project: <http://www.openssl.org>. Accessed 20 Dec 2014
- [16] Kleinrock L (1975) Queueing Systems, Volume I: Theory.
- [17] Bolch G (2006) Queueing Networks and Markov Chains.
- [18] Gross D, Shortle J, Thompson J, and Harris C (2008) Fundamentals of Queueing Theory.
- [19] Cao J, Andersson M, Nyberg C and Kihl M Web Server Performance Modeling Using an M/G/1/K*PS Queue. In: The 10th International Conference on Telecommunications (ICT'03) 23 Feb-1 Mar 2003
- [20] Knesl C, On the sojourn time distribution in a finite capacity processor shared queue. Journal of the ACM. Nov 1993
- [21] Talwar V, Nahrstedt K, and Milojevic D, Modeling remote desktop systems in utility environment with application to QoS management. In: The 11th IFIP/IEEE international conference on Symposium on Integrated Network Management. 2009.
- [22] Uргаonkar B, Pacifici G, Shenoy P, Spreitzer M, and Tantawi A, An Analytical Model for Multi-tier Internet Services and Its Applications. In: ACM SIGMETRICS international conference on Measurement and modeling of computer systems. 6-10 June 2005.

- [23] Han S, Kim N, Choi K, and Kim J, Design of Multi-party Meeting System for Interactive Collaboration. In: The 2nd International Conference on Communication Systems Software and Middleware (COMSWARE07) Jan. 2007.
- [24] Qiu R, Kuhns F, and Cox J, A conference control protocol for highly interactive video-conferencing. In: IEEE Global Telecommunications Conference (GLOCOM02). Nov 2002.
- [25] Park S, Lee S, Kim S, Lee J, and Lee S, A conferencing system for real-time, multiparty, multimedia services. IEEE Transaction on Consumer Electronics. vol 44 issue 3. Aug 1998.

Authors



Chun-Yi Tsai, received his B.S. degree in applied mathematics from the National Chun Hsing University, Taiwan, R.O.C. in 1995, the M.S. degree in computer science & information engineering from the National Sun-Yat-sen University, Taiwan, R.O.C. in 1997, and the Ph.D. degree in computer science & information engineering from the National Taiwan University, Taiwan, R.O.C. in 2009, respectively. Dr. Tsai is currently an assistant professor of the department of computer science & information engineering in National Taitung University, Taiwan, R.O.C.. His research interests focus on computer networks, distributed and parallel computing, multimedia systems, object classifications and recognitions.



Wei-Lun Hung, was born in Taichung, Taiwan, R.O.C. in 1980. He received the B.E. degree in industrial engineering and engineering management from the National Tsing Hua University, Taiwan, R.O.C. in 2005, and the M.S. degree in computer science & information engineering from the National Taiwan University, Taiwan, R.O.C. in 2008. He had also been studying for Ph.D. degree since 2008. His research interests focus on computer vision, embedded system, network security, distributed software architecture and middleware architecture design.

