SISTEM PEMANTAUAN AKTIVITAS PENGGUNA PADA JARINGAN *CLIENT-SERVER*

Siti Khusnul Azifah, Indra Waspada

Jurusan Ilmu Komputer/Informatika, Fakultas Sains dan Matematika, Universitas Diponegoro
e-mail: azifazifa@gmail.com, indrawaspada@gmail.com

Abstrak

Saat ini kebutuhan akan perangkat lunak yang mampu memantau dan menangkap tampilan layar aktivitas pengguna komputer sangat dibutuhkan. Perangkat lunak tersebut penting untuk memastikan apakah komputer benar-benar digunakan untuk bekerja atau hanya untuk melakukan aktivitas lain yang tidak penting. Saat ini, kebanyakan aplikasi pemantau hanya dapat memantau satu desktop dalam satu waktu. Penelitian ini membangun sebuah sistem pemantauan satu arah. Sistem ini menggunakan *Virtual Network Computing* (VNC) untuk melakukan pemantauan secara *realtime* dan *Remote Method Invocation* (RMI) untuk melakukan pemanggilan *method* yang dibutuhkan untuk pemanggilan *screen capture*. Sistem ini diimplementasikan guna memfasilitasi pemantauan satu arah aktivitas pengguna komputer, khususnya di perusahaan, sekolah, dan laboratorium komputer.

Kata Kunci: *Multi*-Desktop, *Virtual Network Computing* (VNC), *Remote Method Invocation* (RMI).

1. PENDAHULUAN

Saat ini kebutuhan akan perangkat lunak yang mampu mendeteksi dan memantau perilaku pengguna komputer semakin meningkat. Pemantauan ini bertujuan untuk mengetahui apakah mereka menggunakan komputer untuk bermain atau bekerja. Hal ini terjadi di berbagai konteks kehidupan, diantaranya adalah di perusahaan, sekolah, dan laboratorium komputer. Pemantauan penggunaan komputer di perusahaan diperlukan untuk memastikan bahwa pegawai memanfaatkan komputer perusahaan untuk bekerja, bukan bermain. Pemantauan penerapan sistem komputerisasi di sekolah diperlukan untuk mengetahui hasil belajar siswa dan mengevaluasi hasil tersebut. Sedangkan pemantauan penggunaan komputer di laboratorium komputer diperlukan untuk memastikan bahwa komputer di laboratorium tersebut digunakan dengan semestinya.

Sebelumnya menurut jurnal Aitichai Jitbanyud yang berjudul *The System of Powerful Computer Laboratory Class via Socket Programming*, telah dibuat aplikasi untuk melakukan pemantauan (*monitoring*). Memanfaatkan *socket programming* untuk meningkatkan kinerja sistem pembelajaran. Pemantauan (*monitoring*) ini bersifat dua arah, baik komputer siswa maupun komputer guru (komputer *control*) memiliki andil dalam jalannya sistem ini. Perbedaannya, komputer guru memiliki akses kontrol terhadap komputer siswa, tidak untuk sebaliknya (Aitichai Jitbanyud, 2010).

Selain itu, menurut jurnal Ajit Kotkar yang berjudul *Android Based Remote Desktop Client*, pemantauan (*monitoring*) juga dapat dilakukan melalui *device* berbasis Android, yaitu dengan memanfaatkan *Virtual Network Computing* (VNC). Pemantauan menggunakan aplikasi ini dilakukan melalui jaringan intranet. Sebuah *viewer* diaplikasikan pada ponsel yang memungkinkan pengguna untuk melihat dan memanipulasi sistem *remote desktop* seperti *MS Windows*. Desktop yang akan diakses harus menjalankan aplikasi *server* dan harus terpasang ke jaringan. Sebuah *proxy* digunakan untuk mengirim gambar dari desktop ke ponsel, untuk mengubah perangkat yang berbeda, untuk menekan lalu lintas jaringan, dan untuk mendukung pemulihan dari pemutusan terjadwal (Ajit Kotkar, 2013).

Kelebihan dari aplikasi belajar mengajar di atas adalah dua pengguna (guru dan siswa) dapat saling berkomunikasi secara *realtime*. Sehingga aplikasi ini mampu melakukan *controlling*, *teaching*, dan *evaluating* terhadap hasil belajar siswa. Namun dalam aplikasi ini *screenshot* layar siswa dikirimkan oleh siswa kepada guru, sehingga tidak ada validasi bahwa *screenshot* tersebut belum dimanipulasi atau diubah dari kondisi sebenarnya oleh siswa. Aplikasi berbasis Android pada paragraf sebelumnya memiliki kelebihan dapat dilakukan secara *mobile* sehingga lebih praktis dan fleksibel untuk dilakukan di manapun. Namun dikarenakan desktop *viewer* merupakan *smartphone*, aplikasi ini hanya memungkinkan untuk memantau satu desktop dalam satu waktu. Mempertimbangkan dua kasus tersebut, penulis memiliki gagasan untuk membuat sebuah sistem pemantau dimana *client* memiliki kendali penuh terhadap sistem. *Screenshot* dikirimkan dari komputer *server* ke komputer *client* dimana kendali dilakukan dari komputer *client* untuk meningkatkan validasi. Dan pemantauan dapat dilakukan pada beberapa desktop dalam satu waktu.

Bahasa pemrograman *Java* telah menyediakan protokol untuk memungkinkan suatu perangkat lunak dapat digunakan untuk memantau komputer lain, lebih tepatnya untuk dapat berkomunikasi dengan

komputer lain. Untuk dapat berkomunikasi dalam jaringan komputer, terdapat beberapa cara, diantaranya: *socket, remote procedure calls* (RPC s), dan RMI. *Socket* merupakan titik akhir untuk komunikasi melalui jaringan. *Remote procedure calls* (RPC s) merupakan bentuk yang paling umum dari *remote service* yang didesain sebagai jalan untuk mekanisme *procedure-call* abstrak untuk digunakan antara sistem dengan koneksi jaringan. Dan *Java remote method invocation* (RMI) adalah fitur Java yang mirip dengan RPCs. RMI memungkinkan suatu *thread* untuk memanggil metode pada *remote* objek (Graba, 2007).

Virtual Network Computing (VNC) adalah desktop sistem *sharing* grafis yang menggunakan protokol RFB untuk kontrol jarak jauh komputer lain. VNC mentransmisikan pergerakan *keyboard* dan *mouse* dari satu komputer ke komputer lain, menyampaikan layar grafis *update* kembali ke arah lain, melalui jaringan (Roebuck, 2011).

Dalam aplikasi pemantauan (*monitoring*) ini *screen capture* digunakan sebagai bukti adanya penyalahgunaan penggunaan komputer. Hal ini sesuai dengan Pasal 5 ayat 1 UU No. 11 Tahun 2008 tentang Informasi dan Transaksi Elektronik ("UU ITE"), yang berbunyi "Informasi elektronik dan/atau dokumen elektronik dan/atau hasil cetaknya merupakan alat bukti hukum yang sah" (UU ITE, 2008).

Penelitian ini akan membuat sebuah sistem untuk memantau aktivitas pengguna dengan *multi desktop viewer* pada jaringan *client-server* yang akan dikombinasikan dengan *remote screen capture.* Selain itu, penulis juga memanfaatkan VNC untuk melakukan pemantauan secara *realtime.* Aplikasi ini nantinya dapat diterapkan di beberapa tempat, seperti perusahaan, sekolah, atau laboratorium komputer.

2. TINJAUAN PUSTAKA

Virtual Network Computing

Virtual Network Computing (VNC) adalah desktop sistem *sharing* grafis yang menggunakan protokol RFB untuk kontrol jarak jauh komputer lain (Roebuck, 2011). VNC mentransmisikan pergerakan *keyboard* dan *mouse* dari satu komputer ke komputer lain, menyampaikan layar grafis *update* ke arah lain, melalui jaringan.

VNC merupakan platform penampil independen. VNC dapat terhubung ke server VNC pada sistem operasi yang sama atau berbeda. VNC terdiri dari *client* dan server VNC, dapat diimplementasikan pada berbagai sistem operasi berbasis GUI dan bahasa pemrograman *Java*. Beberapa *client* dapat terhubung ke satu *server* VNC pada waktu yang sama. Hal ini dapat dimanfaatkan untuk mengakses file di komputer kerja dari komputer di rumah, atau sebaliknya.

Remote Framebuffer Protocol

Protokol *remote framebuffer* (RFB) adalah spesifikasi informasi dari IETF (RFC 6143). Meskipun bukan merupakan standar resmi, RFB secara luas digunakan dan ada banyak implementasi *interoperable*. RFC 6143 telah berusia lebih dari satu dekade dan telah direvisi beberapa kali.

Framebuffer merupakan array yang berisi semua nilai *pixel* yang ditampilkan oleh sistem komputer grafis, dan merupakan model umum terendah denominator dalam sebuah komputer desktop (Vanessa Wang, 2013). Hal tersebut menjadikan RFB sebagai cara untuk mengakses *framebuffer* jarak jauh. Untuk sistem dengan *keyboard, mouse*, dan desktop, dimungkinkan untuk mengaksesnya menggunakan RFB.

Protokol umumnya berorientasi pada data biner atau string teks. Protokol biner dapat lebih selaras dibandingkan protokol berorientasi teks, lebih rapi dan alami dalam menanamkan struktur data biner yang berubah-ubah seperti gambar, audio dan video. Protokol biner dimaksudkan untuk dibaca oleh mesin bukan manusia, dan dapat mengoptimalkan struktur data yang akan dikirim dalam bentuk apapun untuk mempertahankan efisiensi.

Protokol berorientasi teks seperti STOMP dan XMPP cenderung relatif lebih besar untuk mengirimkan pesan pada *wire* dan dengan demikian, lebih mahal untuk mengurai bila dibandingkan dengan protokol biner. Namun, protokol berorientasi teks dapat diterapkan pada hampir semua bahasa, dapat dibaca oleh manusia, dan memiliki panjang field variable yang fleksibel. Sementara protokol biner dapat menjadi cara yang lebih efisien untuk mengangkut data, protokol berorientasi teks dapat memberikan fleksibilitas, dan lebih mudah untuk diterapkan dan dikembangkan.

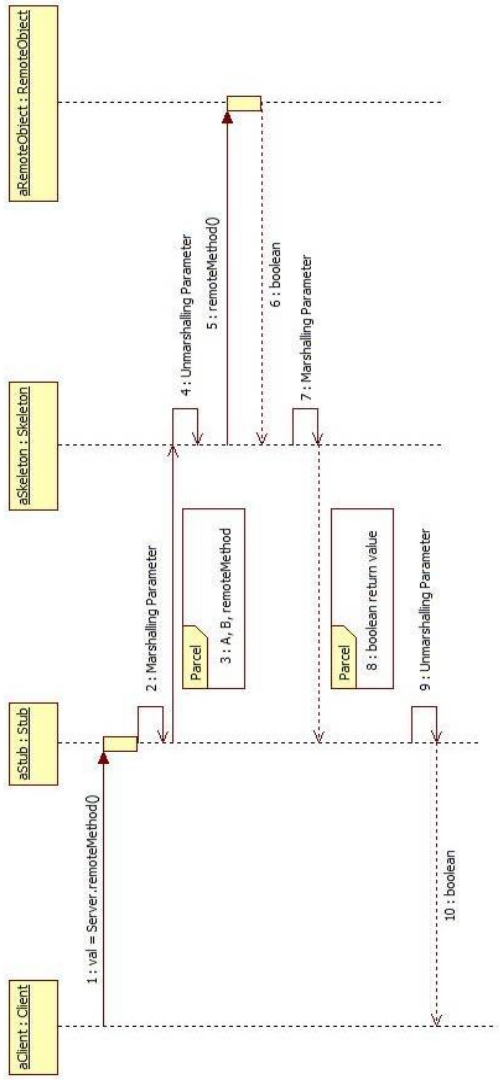
RFB adalah protokol biner yang mentransmisikan data citra biner. Data dapat dikompresi dan dapat dialirkan ke dan dari *server* dengan *update* frekuensi yang sangat tinggi (Vanessa Wang, 2013). Data gambar dapat dialirkan pada frekuensi tinggi dari *server*; demikian pula, *client* dapat menghasilkan aliran peristiwa input yang disebabkan oleh pengguna menggerakkan *mouse* dan menekan tombol. Peristiwa input ini secara kompak dikodekan dalam format biner yang mengambil sangat sedikit *byte* untuk pengiriman.

Remote Method Invocation

Remote method invocation (RMI) merupakan fitur *Java* yang memungkinkan suatu *thread* untuk memanggil *method* pada objek *remote*. Suatu objek disebut objek *remote* jika objek tersebut berada pada *Java virtual machine* (JVM) yang berbeda (Abraham Silberschatz, 2010). Objek *remote* tersebut dapat berada pada JVM lain dalam satu komputer atau dalam satu *remote host* yang terhubung melalui jaringan.

Untuk membuat *remote methods* transparan baik di sisi *client* maupun *server*, RMI mengimplementasikan objek *remote* menggunakan *stubs* dan *skeletons*. *Stub* merupakan suatu *proxy*

untuk objek *remote*, berada pada sisi *client.* Ketika *client* memanggil *remote method*, *stub* pada objek *remote* dipanggil. *Stub* pada sisi *client* ini berfungsi untuk membuat *parcel* berdasarkan nama *method* yang dipanggil pada *server* dan melakukan *marshall* parameter *method* tersebut (Abraham Silberschatz, 2010). Hanya tipe primitif dan jenis-jenis referensi yang mengimplementasikan *interface Serializable* dapat digunakan sebagai parameter (*serializing* parameter ini disebut *marshalling*) (Graba, 2007). *Stub* kemudian mengirim *parcel* ke *server*, dan diterima oleh *skeleton* pada objek *remote* (Abraham Silberschatz, 2010). Setelah menerima aliran *byte*, *skeleton* mengubah *stream* menjadi *method call* sesungguhnya dan parameter terkait (*deserialization* parameter yang disebut sebagai *unmarshalling*) (Graba, 2007). *Skeleton* berfungsi untuk melakukan *unmarshall* parameter dan memanggil *methods* pada *server*. *Skeleton* kemudian melakukan *marshall return value* (atau *exception*) menjadi *parcel* dan mengembalikannya pada *client. Stub* melakukan *unmarshall return value* dan melanjutkannya ke *client.* Proses di atas diilustrasikan pada Gambar 1 di bawah ini (Abraham Silberschatz, 2010).



Gambar 1**.** *Marshalling Parameters*

Package yang digunakan dalam implementasi aplikasi *client-server* RMI adalah *java.rmi*, *java.rmi.server*, dan *java.rmi.registry*, meskipun hanya dua pertama yang dibutuhkan untuk digunakan secara eksplisit. Langkah-langkah dasarnya tercantum di bawah ini (Graba, 2007) :

1. Membuat *interface*
   Interface ini harus mengimpor paket *java.rmi* dan harus *extend interface remote*, yang (seperti *Serializable*) adalah '*tagging*' *interface* yang tidak mengandung *method*. Definisi *interface* untuk contoh ini harus menentukan penandaan untuk *method* getGreeting, yang akan dibuat *available* untuk *client*. *Method* ini harus mendeklarasikan bahwa *interface* tersebut *throw* RemoteException.

2. Menentukan sebuah kelas yang mengimplementasikan *interface* (langkah 1)
   File implementasi harus mengimpor paket *java.rmi* dan *java.rmi.server*. Kelas implementasi harus *extend* kelas RemoteObject atau salah satu dari *subclass* RemoteObject. Pada praktiknya, sebagian besar implementasi *extend subclass* UnicastRemoteObject, karena kelas ini mendukung komunikasi *point-to-point* menggunakan TCP *stream*. Kelas implementasi juga harus mengimplementasikan *interface Hello*, tentunya, dengan menyediakan file *executable* untuk *method single interface* getGreeting. Selain itu, harus dibuat konstruktor untuk objek implementasi (bahkan jika konstruktor tersebut kosong). Seperti *method*(*s*) yang dideklarasikan pada *interface*, konstruktor ini harus menyatakan *throws* RemoteException.

3. Membuat proses *server*
   Server menciptakan objek dari kelas implementasi di atas dan melakukan register dengan layanan penamaan disebut *registry*. Hal ini dilakukan dengan menggunakan *method* statis rebind kelas Naming (dari paket *java.rmi*). *Method* ini membutuhkan dua argumen:
   - String yang menyimpan nama objek *remote* sebagai URL dengan protokol RMI;
   - Referensi ke objek *remote* (sebagai argumen tipe *Remote*).
     Method ini membentuk koneksi antara nama objek dan referensi. Klien kemudian dapat menggunakan nama dari objek *remote* untuk mengambil referensi ke objek melalui *registry*.

   String URL, selain menentukan protokol RMI dan nama objek, juga menentukan nama *host* mesin objek *remote*. Untuk mudahnya, gunakan localhost (yang merupakan *host default* RMI). Port *default* untuk RMI adalah 1099, port tersebut dapat diubah sesuai keinginan. Kode untuk proses

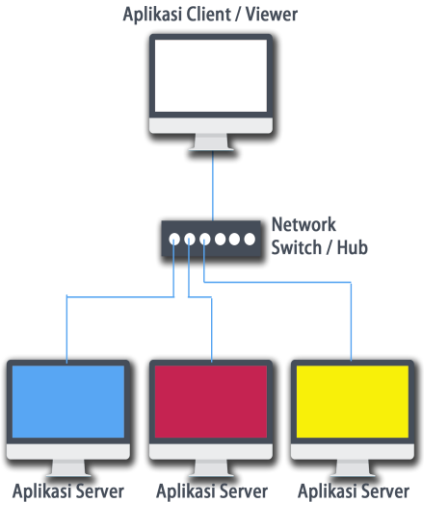
server ditunjukkan di bawah dan hanya berisi satu *method* utama. Untuk memenuhi berbagai jenis pengecualian yang mungkin dihasilkan, *method* ini menyatakan *throws Exception*.

4. Membuat proses *client*
   Client mendapatkan referensi ke objek *remote* dari registri. Hal ini dilakukan dengan menggunakan *method* lookup kelas Naming, menyediakan argumen untuk *method* ini dengan URL yang sama dimana *server* melakukan *binding* objek referensi untuk nama objek pada registri. Karena lookup mengembalikan referensi *Remote*, referensi ini harus *typecast* menjadi referensi *Hello* (bukan referensi *HelloImpl*). Setelah referensi *Hello* diperoleh, dapat digunakan untuk memanggil *method* soliter yang tersedia pada *interface*.

Beberapa penulis memilih untuk menggabungkan implementasi dan *server* ke salah satu kelas. Hal tersebut bersifat *optional* untuk dilakukan.

Arsitektur Sistem

Gambar 2 di bawah ini merupakan arsitektur sistem pemantauan yang akan dibangun. Dalam sistem ini terdapat sebuah komputer *client* dan beberapa komputer *server* yang berada dalam satu jaringan.



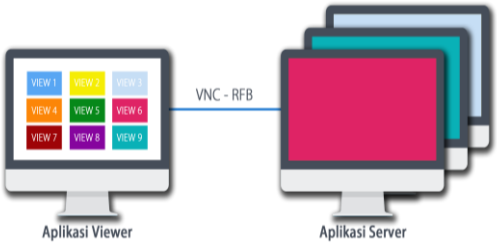
Gambar. 2. Arsitektur Sistem Pemantauan Aktivitas Pengguna Pada Jaringan *Client-Server*

Komputer *client* berisi aplikasi *viewer/ client* bertugas mengendalikan kerja seluruh sistem ini, mulai dari menentukan aplikasi *server* yang ingin dipanggil, melakukan pemanggilan, dan menampilkan hasil dari *screen capture* aplikasi *server*. Sedangkan Komputer *server* berisi aplikasi *server* bertugas melakukan *screen capture* ketika aplikasi *server* dipanggil oleh aplikasi *client* dan mengirimkan hasil *screen capture* tersebut ke aplikasi *client* melalui *socket*.

Garis Besar Penyelesaian Masalah

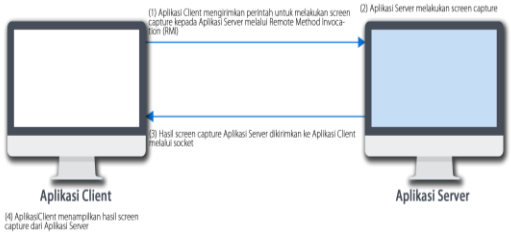
Sistem pemantauan ini nantinya akan dibangun menggunakan bahasa pemrograman *Java*. Dalam sistem ini terdapat dua fungsionel utama yaitu melakukan pemantauan *realtime* menggunakan VNC dan pengiriman *screen capture* menggunakan RMI dan *Socket Programming*.

Dalam fungsional pertama (pemantauan *realtime*) komputer *client* berisi aplikasi *viewer* dan komputer *server* berisi aplikasi *server*. Tugas aplikasi *viewer* adalah menampilkan desktop-desktop komputer *server* yang sedang dipantau dan pergerakannya secara *realtime.* Ilustrasinya seperti pada Gambar 3 berikut ini.



Gambar. 3. Memantau Komputer Lain Menggunakan VNC

Sedangkan fungsional kedua (pengiriman *screen capture*), sebuah komputer admin berisi aplikasi *client* dan komputer *server* berisi aplikasi *server*. Komputer-komputer tersebut berada dalam satu jaringan (lihat Gambar 2). Gambar 4 berikut ini merupakan alur sistem pengiriman *screen capture* yang akan dibuat.



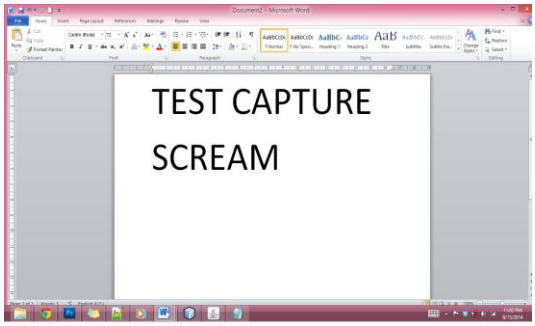
Gambar. 4. Alur Sistem Pengiriman *Screen Capture*

Langkah pertama adalah aplikasi *client* mengirimkan perintah untuk melakukan *screen capture* kepada aplikasi *server* melalui *remote method invocation* (RMI). Hal ini dapat dilakukan baik pada satu maupun banyak aplikasi *server.* Kemudian aplikasi *server* (yang dipanggil oleh aplikasi *client*) melakukan *screen capture* pada layar yang sedang aktif. Hasil dari *screen capture* kemudian dikirim ke aplikasi *client* melalui *socket.* Kemudian hasil *screen capture* dari aplikasi *server* tersebut disimpan dan dapat ditampilkan pada aplikasi *client*.

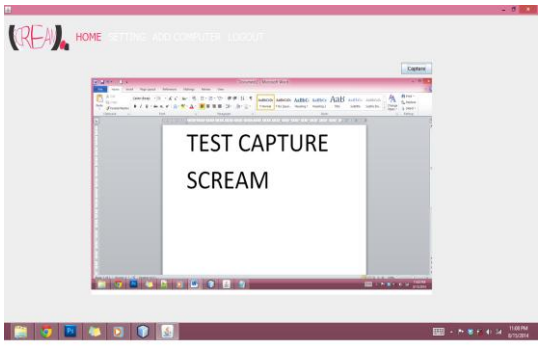
Hasil Eksperimen

Implementasi dilakukan dengan dua cara yaitu dengan melakukan pemantauan secara *realtime* dan

mengambil *capture screen* dari komputer *server* dan kemudian dikirimkan dan disimpan di komputer *client.*

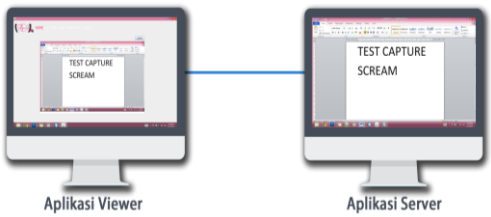


Gambar. 5. Desktop Komputer *Server*



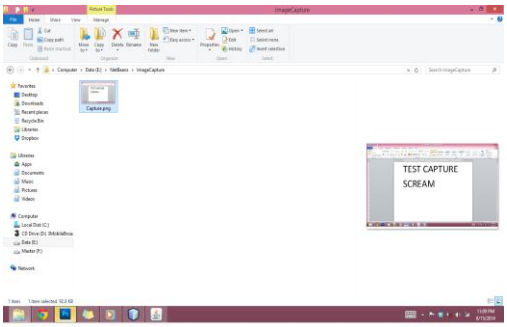
Gambar. 6. Halaman *Home* Menampilkan Tampilan Desktop Komputer *Server*

Dari Gambar 6 terlihat bahwa desktop komputer *server* dapat dipantau dari desktop komputer *client*. Ilustrasinya ditampilkan pada Gambar 7 berikut ini. Hal ini dilakukan dengan menggunakan plugin *Virtual Network Computing* (VNC).



Gambar. 7. Ilustrasi Pemantauan Secara *Realtime*

Gambar 8 merupakan implementasi dari pengambilan *capture screen* desktop komputer *server*. Prosesnya adalah pengguna menekan tombol *capture* pada halaman *Home* (lihat Gambar 6), kemudian komputer *server* akan melakukan *capture screen* desktop dan mengirimkannya ke komputer *client*. Hasil *capture* tersebut kemudian disimpan dalan folder di komputer *client* (lihat Gambar 8).



Gambar. 8. Folder Hasil *Capture Screen* Desktop Komputer *Server*

3. KESIMPULAN

Berdasarkan pada hasil pengujian pada tahap implementasi, maka dapat disimpulkan beberapa kesimpulan sebagai berikut:

1. Salah satu kelemahan VNC adalah jika pada komputer *server*, *server* yang aktif adalah *user root*, maka *client* yang melakukan *remote login* akan masuk sebagai *root.* Sehingga bisa mengakses file-file *system* yang ada di komputer *server*. Untuk menjaga keamanan komputer *server* sebaiknya tidak login di *server sebagai root* karena dapat membahayakan komputer *server*.
2. Pengambilan *capture screen* akan bertambah lama bila melibatkan lebih banyak komputer.

Saran

Untuk penelitian selanjutnya dapat menambahkan jumlah komputer yang dipantau dan fitur-fitur tambahan yang dibutuhkan, sehingga dapat diimplementasikan dalam lingkup yang lebih besar.

4. DAFTAR PUSTAKA

[1] Graba, J., *An Introduction to Network Programming with Java*. USA: Springer, 2007.

[2] Jitbanyud, A. & Toadithep, N., The System of Powerful Computer Laboratory Class via Socket Programming, *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on (Volume:4)*, 2010.

[3] Kotkar, A., Alok Nalawade, A. & Gawas, Android Based Remote Desktop Client, *International Journal of Innovative Research in Computer and Communication Engineering (Volume: 1)*, 2013.

[4] Republik Indonesia. 2008. Undang-Undang Nomor 11 Tahun 2008 Tentang Informasi dan Transaksi Elektronik. Lembaran Negara RI Tahun 2008, No. 4843. Sekretariat Negara. Jakarta.

[5] Roebuck, K., *Virtual Network Computing (VNC): High-impact Strategies*, Sidney: Emereo Pty Limited, 2011.

[6] Wang, V., Salim, F. & Moskovits, P., *The Definitive Guide to HTML5 WebSocket*, New York: Apress, 2013.