Jurnal SimanteC
Vol. 4, No. 3 Juni 2015 ISSN 2088-2130

RANCANG BANGUN SISTEM *INSTANT MESSAGING* TERDISTRIBUSI

H u s n i

Program Studi Teknik Informatika, Fakultas Teknik, Universitas Trunojoyo Madura

Jl. Telang Raya Kamal, Bangkalan, Jawa Timur

husni@if.trunojoyo.ac.id

Abstrak

Instant messaging adalah salah satu jenis aplikasi *online* untuk komunikasi *real-time* yang semakin banyak digunakan di Internet. Vendor-vendor besar terlibat dalam penyediaan layanan Internet ini, seperti Microsoft, Yahoo dan Google. Fitur yang ditawarkan juga semakin lengkap sehingga komunikasi langsung antar pengguna Internet dengan berbagai medianya sudah terwujud. Namun berbagai teknologi yang dimunculkan tersebut tidak terbuka sehingga sulit bagi pemrogram aplikasi atau peneliti awal mengetahui cara kerja dari sistem messaging. Kami telah merancang dan membangun suatu prototipe sistem *instant messaging* (*messenger*) sederhana dengan server tunggal dan kemudian diperbaikinya dengan mewujudkan beberapa server terdistribusi yang saling berkoordinasi untuk meningkatkan *fault-tolerance* dan *availability*. Implementasi dari dua pendekatan ini telah mampu menjawab bagaimana proses pembangunan aplikasi jaringan komputer yang memanfaatkan lapisan *transport* dan *network* pada model referensi OSI. Sentuhan langsung dengan level sistem memungkinkan kita mengetahui secara lebih jelas dan lengkap bagaimana proses komunikasi sesungguhnya yang terjadi pada suatu sistem *instant messaging* dari pendekatan yang diterapkan.

Kata kunci: *instant messaging, messenger, sistem terdistribusi, pemrograman jaringan*

Abstract

Instant messaging is one type of online applications for real-time communications that are increasingly being used on the Internet. Major vendors involved in the provision of the Internet services, such as Microsoft, Yahoo and Google. The features offered are also more complete so that direct communication between Internet users with a variety of media already materialized. However, various technologies that emerged were not open so difficult for the application programmer or early researchers to know the workings of the messaging system. We have designed and built a prototype of simple instant messaging system (Messenger) with a single server and then repaired with realizing several distributed servers which coordinate with each other to increase the fault-tolerance and availability. The implementation of these two approaches has been able to answer how the process of building computer networks applications that take advantage of network and transport layers in the OSI reference model. Touch directly with the level of the system allows us to know more clearly and fully how the actual communication process that occurs in an instant messaging system of the approach applied.

Keywords: *instant messaging, messenger, distributed system, network programming*

1. Pendahuluan

Instant messaging merupakan salah satu perangkat yang sangat banyak digunakan saat ini [1]. Menurut Wikipedia [2], *instant messaging* adalah teknologi yang memungkinkan komunikasi *real-time* berbasis teks antara dua atau lebih partisipan memanfaatkan internet atau intranet. Server yang menyediakan layanan *messaging* biasa dinamakan *Messenger*. Beberapa sistem memungkinkan pengiriman pesan ke orang-orang yang tidak sedang *login* (*offline message*) sehingga sedikit menyamai fungsi dari layanan *messaging* berbentuk email. Banyak layanan tambahan dijadikan fitur dari layanan ini seperti *group chatting*, *conference* yang melibatkan suara dan video, *file transfer* dan pencatatan (*logging*) percakapan [3]. Bahkan ada *messenger* yang mengakomodasi pemakaian *web-cam* atau percakapan langsung melalui Internet.

Table 1. Deskripsi fungsional protokol Messenger

No.	Nama	Deskripsi
1	Login, Registrasi	Menerima permintaan login dan mendaftarkan pengguna dalam sistem lengkap
2	Mengambil daftar pengguna	Mendapatkan daftar pengguna "online" yang dapat menerima pesan
3	Mendapatkan pesan (inbox)	Mendapatkan daftar semua pesan dari inbox
4	Membaca pesan	Mendapatkan pesan tertentu dari inbox dan menampilkan nya kepada pengguna
5	Mengirim pesan	Mengirimkan suatu pesan kepada pengguna tertentu
6	Menghapus pesan	Menghapus pesan tertentu
7	Logout, *Unregistered*	Logout dan menghapus pengguna dari sistem lengkap.

Secara umum, sistem *intant messaging* terdiri dari satu server (*messenger*) dan banyak *client* [4]. Seorang pengguna (*client*) dapat mengirimkan pesan teks kepada satu atau lebih pengguna lain. Beberapa sistem menyediakan fasilitas pengelompokan pengguna melalui *channel*, misalnya berdasarkan negara atau sekolah sehingga pesan dapat diterima oleh setiap anggota dalam *channel* tersebut. Pengguna juga dapat membentuk *channel* sendiri dan kemudian mengundang pengguna lain untuk bergabung.

Sistem *instant messaging* yang lebih baru mengharuskan penggunanya mendaftar terlebih dahulu, biasanya memanfaatkan layanan email dari penyedia yang sama. Contohnya adalah Yahoo Messenger, Google Messaging, dan Microsoft Network. Registrasi tersebut memungkinkan berbagai layanan tambahan dapat diaplikasikan, seperti *offline messaging*, pemberitahuan melalui email, pengaturan *profile*, dan notifikasi email masuk. Fungsi utama dari aplikasi messenger dan pertukaran informasi antara client dan server diperlihatkan pada tabel 1 [5].

Tabel 2. Perbandingan beberapa sistem *instant messenger* popular.

Messenger	Keunggulan	Kelemahan
Jabber	Arsitektur server terdesentralis	Belum lengkap Tidak

	asi Berbasis XML open source	populer
ICQ	Mengirimkan pesan offline Menyediakan fungsionalitas unggulan Arsitektur server tersentralisasi	Bukan protokol open source Aspek keamanan Komunikasi sinkron/lambat Banyak operasi di sisi client
MSN Messenger	Protokol asinkron (SIMPLE) Fungsi banyak Arsitektur server tersentralisasi	Bukan protokol open source Terintegrasi dengan Windows Target virus
Yahoo! Messenger	Arsitektur server tersentralisasi Terintegrasi sangat baik dengan web Ruang chat yang menarik	Bukan protokol open source Banyak selingan: game, iklan, radio Keamanan lemah

Zheng [6] membandingkan 4 sistem *instant messenger* yang banyak digunakan dan rangkumannya diperlihatkan pada tabel 2.

Peningkatan jumlah pengguna *instant messaging*, seiring meningkatnya jumlah pengguna Internet, mengakibatkan sistem *messaging* satu server tidak selalu dapat diandalkan. Penyedia harus Pada sistem server terdistribusi ini, pengguna akan tersambung ke server *messenger* terdekat dan dapat berkomunikasi dengan pengguna yang tersambung pada server lain. Jika salah satu server mati (*down*) maka pengguna dapat beralih atau dialihkan ke server lain [7].

menyediakan beberapa server tambahan agar kinerja sistem secara keseluruhan tetap stabil. Akhirnya muncul inisiatif memuat banyak server yang tersebar.

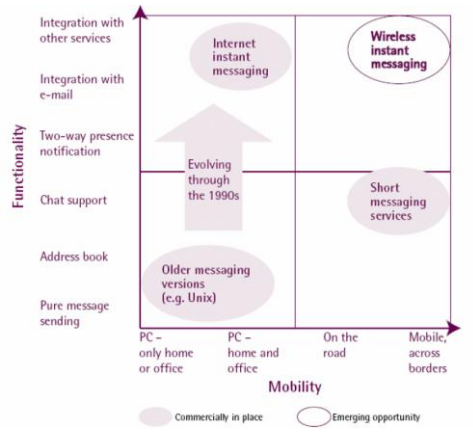
Menurut Lundgren [4] pesan-pesan dalam sistem IM dapat diteruskan melalui server khusus (*dedicated*) atau secara langsung antar pihak-pihak yang berkomunikasi (peer-to-peer), tergantung dari arsitektur yang dipilih. Tabel 3 memperlihatkan perbandingan 6 Messenger yang umum digunakan saat ini.

Tabel 3. Perbandingan software messenger popular

Protokol	**Registrasi /Lookup**	**Pertukaran Pesan**
AIM/OSCAR	CS	CS
ICQ	CS	P2P
IRC (DCC)	Tersebar	CS/P2P
Jabber	CS	P2P
MSN IM	CS	CS
Yahoo	CS	CS

Catatan: CS = Centralized Server, P2P = peer-to-peer

Di lihat dari sisi sejarah, IM ternyata berkembang dari program talk yang hadir di sistem Unix. Gambar 1 memperlihatkan perkembangan aplikasi IM tersebut.

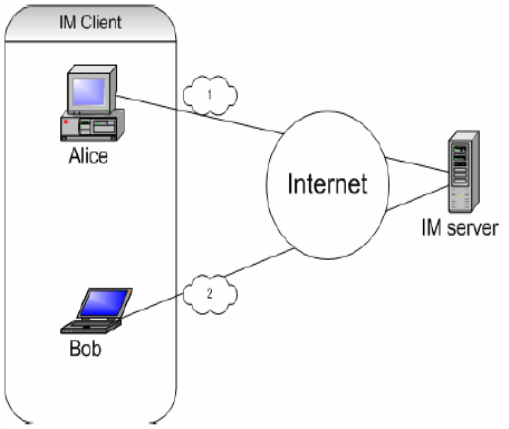


Gambar 1. Perkembangan *Instant Messaging* dimulai dari program talk di Unix [3]

Tulisan ini tidak mengusulkan rancangan atau implementasi terbaru di bidang *messenger*. Kami telah merancang dan membuat prototipe dari sistem *messenger* tunggal yang menghadirkan masalah saat jumlah pengguna yang perlu ditangani semakin banyak. Pendekatan *messenger* server terdistribusi kemudian dipilih sebagai solusi karena mampu meningkatkan *fault-tolerence* dan *availability* [8] dari layanan *instant messaging* tersebut. Fokus tulisan ini adalah penjabaran cara kerja dari kedua pendekatan yang diimplementasikan beserta kelebihan dan kekurangan yang ditemui ketika pengembangan sistem dilakukan. Kami berharap pengalaman ini dapat menjadi acuan bagi sistem programmer dalam membangun aplikasi komputer yang bersentuhan dengan level sistem terutama layer transport dan network dari model OSI.

2. Server Messenger Tunggal

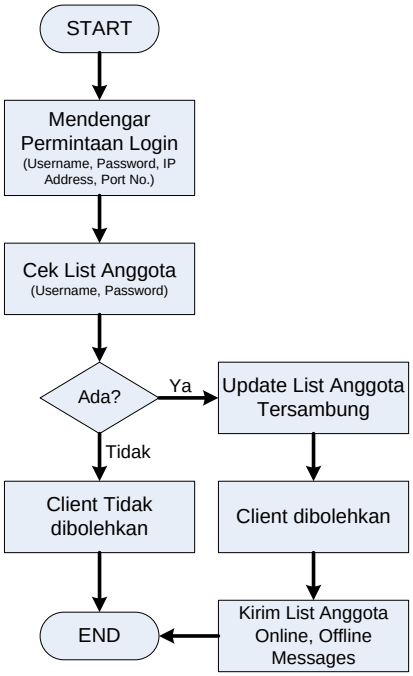
Sistem sederhana ini terdiri dari satu server *instant messenger* (IM) dan beberapa client. Client dapat mengirimkan pesan kepada satu atau lebih pengguna. Pesan-pesan dari pengguna akan disimpan di server sebelum diteruskan ke tujuan (gambar 2). Server mengharuskan pengguna terdaftar dan login terlebih dahulu sebelum dapat mengirimkan pesan. Semua komunikasi teks antar pengguna melewati dan diatur oleh server, bertindak sebagai *central of control*. Secara garis besar, proses login yang ditangani oleh server diperlihatkan pada gambar 2.



1. Pesan dikirimkan terlebih dahulu ke server IM
2. Server IM mengirimkan pesan ke pengguna yang dimaksudkan.

Berikut ini adalah beberapa fitur dasar yang disediakan oleh *single messenger server*:

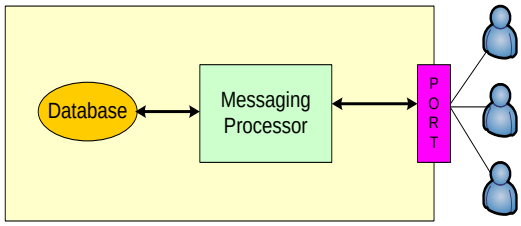
1. Pengguna hanya dapat mengirimkan pesan kepada satu penerima atau semua pengguna sekaligus, tidak di antaranya (misalnya dua atau tiga penerima pada satu waktu pengiriman).
2. Penerima pesan tidak harus sedang *login* tetapi harus telah menjadi anggota. Pesan-pesan yang diterima akan disimpan oleh server kemudian diteruskan langsung kepada pengguna (IP Address dan nomor Port tertentu) jika sedang tersambung ke server. Jika penerima sedang *off* maka pesan hanya akan disimpan. Pesan tersebut akan dikirimkan ke penerima saat *login* berikutnya sebagai *offline messages*. Fitur ini tidak berlaku bagi pesan yang dikirimkan ke semua pengguna karena akan mengakibatkan terlalu banyak *offline message* dan menurunkan kinerja.



Gambar 3. Proses penanganan login pengguna

3. Beberapa pengguna berbeda dapat *login* melalui mesin yang sama, namun harus berbeda *instance* dari program *client*. Jika dua pengguna berada di satu PC maka harus menjalankan 2 program *client*.
4. Server akan memberikan daftar semua pengguna yang sedang tersambung setiap kali ada pengguna yang berhasil login. Pengembangan ke depan perlu membatasi jumlah pengguna yang harus diinformasikan kepada client sesuai dengan daftar teman yang dibuat pengguna.

Arsitektur dari server messenger tunggal ini diperlihatkan pada gambar 4.

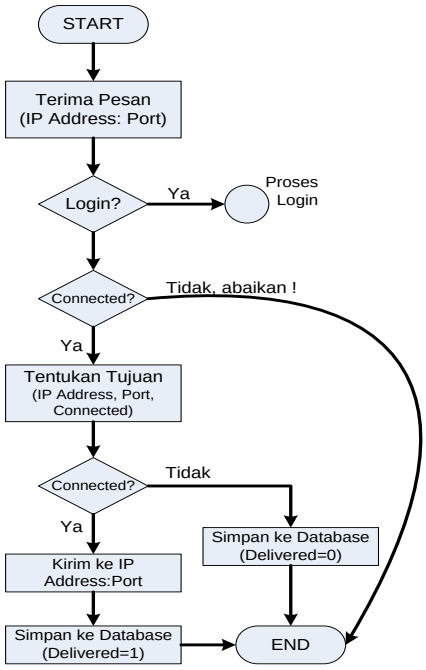


Gambar 4. Arsitektur dari messenger dengan satu server

Dari sisi database, sistem ini setidaknya memerlukan 3 tabel, yaitu:

1. Tabel **User**, menyimpan semua data pengguna yang telah terdaftar. Informasi paling penting pada tabel ini adalah Username dan Password yang akan diperiksa saat pengguna login.
2. Tabel **Connected**. Tabel ini mencatat sejarah koneksi dari pengguna. Informasi penting yang disimpan mencakup Tanggal dan Jam pengguna login, Username, IP Address dan nomor Port yang digunakan oleh program client, dan status tersambung (*Connected*). Jika *Connected* bernilai 1 maka menyatakan bahwa Username tersebut sedang tersambung (aktif, login) dan jika bernilai 0 maka sedang *off*.
3. Tabel **Messages**. Tabel ini menyimpan semua pesan yang diterima oleh server. Field-field pentingnya adalah Tanggal dan Jam pesan diterima, Username pengirim pesan (*Source*), Username penerima pesan (*Destination*), Pesan itu sendiri dan status pengiriman pesan (*Delivered*). Jika Delivered bernilai 1 maka pesan dianggap telah terkirim dan jika bernilai 0 maka dianggap *offline message* dan akan diserahkan ke pengguna saat login berikutnya.

Tugas utama dari server *messenger* adalah *messaging* yaitu menyampaikan pesan dari suatu sumber ke tujuan. Proses ini adalah (gambar 5):



Gambar 5. Proses *messaging*

1. Server menerima pesan teks yang masuk pada *port* tertentu (misalnya 4999). Server tidak mengetahui siapa "Username" (*Source*) yang mengirimkan pesan tersebut. Informasi yang diperoleh hanya IP address dan nomor Port pengirim (dimana program client berjalan) beserta pesan yang akan diteruskan.
2. Server memeriksa apakah pesan tersebut berisi permintaan login atau membangun koneksi. Jika Ya, maka ditangani oleh proses penanganan login yang diperlihatkan pada gambar 3. Jika Tidak, berarti pesan tersebut untuk diteruskan kepada pengguna lain.

3. Server memeriksa apakah pengguna yang mempunyai IP Address dan nomor Port tertentu tersebut sudah login dan terdapat di dalam tabel Connected. Jika belum terdapat di dalam tabel Connected (dengan status Connected = 0) maka pesan tersebut diabaikan. Hanya pesan dari pengguna yang telah login (Connected=1) yang boleh memanfaatkan server *messaging* ini.
4. Jika IP address dan nomor *Port* (pengirim) di atas terdapat di dalam tabel Connected dengan *field* Connected bernilai 1 maka Server memeriksa kembali pesan yang diterima tersebut. Kepada siapakah pesan tersebut ditujukan? Di sini hanya diperoleh Username tujuan (*Destination*), tidak ada IP address dan nomor *Port*-nya. Server memeriksa tabel Connected untuk mengetahui status Connected dari tujuan, sekaligus mendapatkan IP address dan nomor Port-nya.
5. Jika diperoleh Connected =1 maka server segera mengirimkan pesan tersebut ke alamat tujuan tersebut, kemudian menyimpan pesan ke dalam tabel Messages dengan *field* Delivered bernilai 1. Jika Connected dari tujuan bernilai 0 maka message langsung disimpan di dalam tabel Messages dan field Delivered diberi nilai 0.

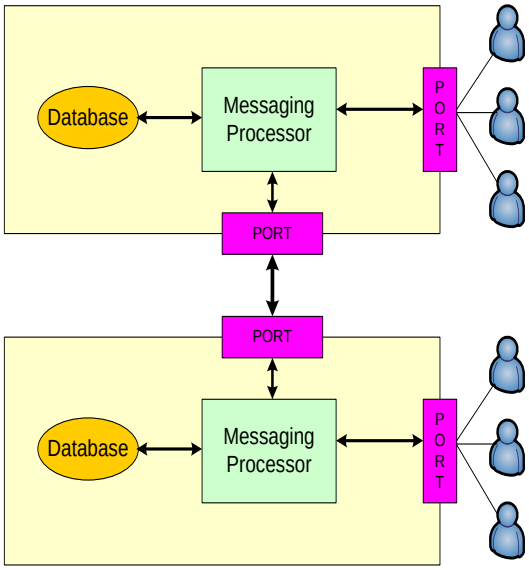
Komunikasi antara server dan client menggunakan socket dari User Datagram Protocol (UDP). Pemilihan terhadap UDP, bukan TCP, didasari pada beberapa pertimbangan, yaitu [9,10]:

1. Proses mengiriman pesan melalui UDP lebih cepat karena tidak melibatkan proses *three ways handshake*.
2. Pemanfaatan TCP memang menghemat *bandwidth* karena koneksi hanya dibangun sekali. Namun pada aplikasi chat atau messaging dimana pengguna tidak sibuk mengirimkan pesan maka koneksi (*channel*, *socket*) tetap terbuka. Jika server membatasi jumlah thread untuk client, misalnya hanya menerima 100 client pada satu waktu maka banyak pengguna yang tidak dapat terkoneksi. Jika server tidak membatasi jumlah pengguna yang dapat login secara simultan maka akan menurunkan kinerja dari server termasuk penggunaan resource yang berlebihan terutama *memory*.
3. UDP bersifat *connectionless*. Koneksi hanya bersifat sementara yaitu saat pesan dikirim. Ini memang memboroskan *bandwidth* karena pesan yang dilewatkan ke jaringan menjadi lebih besar ukurannya. Namun, server dapat menerima banyak pesan pada satu waktu. Karena tidak menyiapkan satu *channel* khusus untuk setiap client maka server harus memeriksa IP address dan nomor *Port* setiap pesan yang masuk, juga harus mencari IP address dan nomor *Port* tujuan.

3. Server Messenger Terdistribusi

Sifat terdistribusi pada sistem *messaging* yang dirancang berarti terdapat lebih dari satu server yang berjalan pada mesin berbeda. Penambahan server ini tidak mengubah cara kerja server tunggal dalam menangani pesan dari dan ke pengguna. Namun, karena telah berada dalam lingkungan terdistribusi maka beberapa fitur harus ditambahkan ke dalam server tunggal sebelumnya sehingga beberapa tujuan dari *distributed system* dapat dicapai [8].

Garis besar arsitektur sistem *messaging* terdistribusi ini diperlihatkan pada gambar 6. Komunikasi antara *client* dan server tetap seperti sebelumnya, yaitu melalui suatu nomor *port* tetap, misalnya 4999. Sedangkan komunikasi atau pertukaran pesan antar server (Inter-Server) melalui nomor *port* yang lain, misalnya 5000. Jadi setiap server menerima pesan dari dua *port* tetap. Sedangkan *client* akan mendapatkan nomor *port* dinamis sesuai dengan mekanisme dari sistem operasi dimana program *client* berjalan.



Gambar 6. Arsitektur sistem *messaging* terdistribusi dengan 2 server

Khusus mengenai database, karena tidak menggunakan *embeded* database, maka suatu server database dapat dimanfaatkan oleh satu atau lebih server *messenger* terutama pada server *messenger* dengan mesin berkinerja menengah. Struktur dari tabel juga mengalami sedikit perubahan untuk mengakomodasi koordinasi antar server. Tabel Connected ditambahkan *field* "Server" untuk mencatat di server mana seorang pengguna terkoneksi. Tidak ada perubahan terhadap tabel Messages. Tabel baru yang diperlukan adalah ServerList. Tabel ini menyimpan daftar server yang beroperasi mencakup IP address, nomor Port, Active dan Key (kunci akses untuk mengamankan server dari akses nakal). Daftar server diurutkan berdasarkan kedekatan akses dari server dimana daftar ini diletakkan.

Beberapa tugas penting dari sistem *messaging* terdistribusi ini adalah

1. Update Server.

   Setiap server memegang daftar server baik yang aktif maupun yang sedang down. Daftar server harus diusahakan sama pada tiap server *messenger*.

   Jika ada suatu server messenger baru, maka administrator harus menyimpan daftar server yang telah ada. Jadi, hanya server baru yang perlu meng*update* daftar server secara total. Berdasarkan daftar server yang telah ada, server baru tersebut mengirimkankan informasi ke server dalam daftar bahwa terdapat server baru yang berjalan pada IP address, nomor *port* dan key tertentu, yaitu dirinya sendiri.

   Server yang menerima informasi server baru tersebut tidak perlu meneruskan informasi itu kepada server lain. Server ini cukup meng*update* informasi lokalnya agar daftar servernya menunjukkan kondisi terkini. Jadi hanya satu server yang mengirimkan informasi *update* ke semua server dalam daftar.

   Secara berkala, misalnya per 5 menit, setiap server harus memeriksa kondisi server lain misalnya dengan memanfaatkan utilitas *ping* atau *trace route*. Jika hasil ping terhadap suatu server memperlihatkan bahwa server "*Unreachable*" maka status dari server di dalam tabel ServerList akan dijadikan 0 (jika aktif bernilai 1) dan tidak akan digunakan dalam proses *update* atau *fault-tolerance*.

2. Update User

   Setiap pengguna harus terdaftar. Pengguna dapat terdaftar di server manapun. Mekanisme *update* user akan segera mengirimkan informasi tentang pengguna tersebut (Username dan Password) kepada server lain. Ini dilakukan untuk menghindari terjadinya Username ganda pada lingkungan terdistribusi. Semua server aktif akan menerima informasi pengguna terbaru tersebut.

3. Update Connected.

   Update terhadap tabel Connected sebaiknya segera dilakukan begitu pengguna dinyatakan berhasil *login* pada suatu server. Informasi dalam tabel ini sangat penting dalam proses *fault-tolerance* dan *load balancing*. Update ini hanya dilakukan untuk baris-baris yang nilai *field* Connected = 1. Jika *field* bernilai 1 berarti pengguna sedang *login* (menggunakan sistem)

dan status ini harus dijaga oleh server semaksimal mungkin sampai pengguna itu menyatakan *disconnect*.

Namun, karena proses koneksi client melalui UDP sedangkan komunikasi antar server melalui TCP maka cara di atas sulit diterapkan. Salah satu solusinya adalah secara berkala setiap server mengirimkan baris-baris tabel Connected terbaru yang *field* Connected bernilai 1, misalnya per menit.

4. Update Messages.

   Update terhadap tabel Message diperlukan terutama untuk mengakomodasi fitur *offline messaging*. Karena itu, hanya baris-baris dengan *field* Delivered = 0 yang dikirimkan ke semua server aktif. Dengan adanya fitur ini, maka ke server manapun pengguna login ia akan mendapatkan *offline message*nya, jika ada. Pengiriman message ini juga dilakukan secara periodik.

5. Load balancing

   Pada sistem *messaging* berbasis UDP memang sulit untuk mengatur keseimbangan beban antar server walaupun sistem ini terlihat *real-time* namun sebetulnya sistem *messaging* termasuk *asynchronous*. Informasi kritis yang dapat digunakan hanya berasal dari tabel Conntected.

   Tabel Connected memperlihatkan daftar pengguna yang aktif beserta di server mana pengguna tersebut *login*. Server dapat meminta client untuk login pada server lain yang mempunyai beban lebih kecil. Namun ini merupakan proses tidak mudah dan belum tentu server yang baru yang dijadikan tempat *login* tersebut akan menguntungkan pengguna.

   Langkah yang paling efektif adalah membatasi jumlah pengguna yang dapat terkoneksi pada suatu server. Setiap server dapat mempunyai nilai berbeda untuk ini. Server dengan spesifikasi tinggi tentu dapat menangani lebih banyak pengguna aktif. Saat server menginginkan *login* (/connect), server memeriksa apakah *quota* untuk dirinya sudah terpenuhi. Jika sudah mencapai batas maka server ini cukup mengatakan kepada *client* bahwa "server penuh" atau "*full*". Informasi ini (*full*) akan digunakan oleh *client* untuk *login* berikutnya.

   Dari DNS server, *client* mengetahui server-server lain yang dapat dihubungi dan melalui *ping* atau *trace route* dapat diperoleh server kedua yang menguntungkan bagi *client*. *Client* mencoba *login* pada server kedua tersebut. Proses penolakan berdasarkan *quota* ini lebih mudah diaplikasi dan tidak membebani server dalam mengatur keseimbangan beban kerja antar server.

6. *Fault tolerance*.

   Salah satu fungsi paling penting dalam sistem terdistribusi adalah *fault tolerance* yaitu bagaimana menjamin agar layanan tetap tersedia, meskipun dengan kinerja sedikit menurun, saat salah satu server mengalami kegagalan dan server lain mengambil alih pekerjaan yang ditangani server yang gagal tersebut.

   Pada sistem *messaging* yang dirancang, *fault tolerance* adalah pengalihan koneksi bagi *client* dari satu server ke server lain. Ada dua solusi yang dapat digunakan, yaitu:

   - Informasi dari server.

     Selain bertukar informasi mengenai *update* di atas, antar server juga perlu melakukan pemeriksaan terhadap server untuk mengetahui adanya server yang mati (*down*) padahal tabel Connected memperlihatkan adanya pengguna yang sedang login. Tool ***ping*** dan ***trace route*** dapat digunakan untuk keperluan ini. Misalnya secara berkala, suatu server memeriksa "*alive*" tidaknya semua server di dalam daftarnya. Jika server ini menemukan adanya server yang

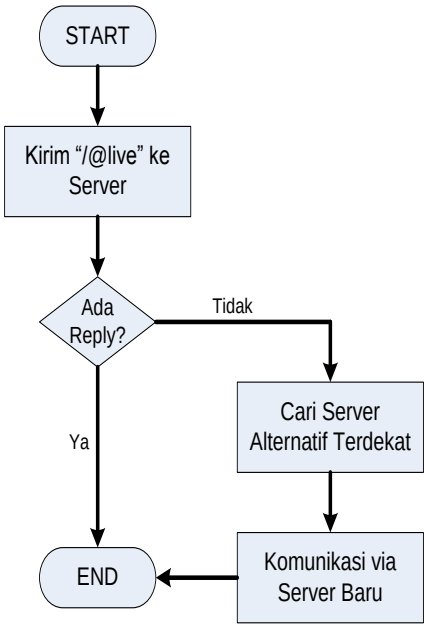
down tetapi berdasarkan isi tabel Connected lokal terdapat pengguna aktif maka server ini segera memberitahukan client (pada IP *address* dan nomor *port* tertentu) untuk segera mengalihkan komunikasi teks ke dirinya.

Isi tabel Connected juga diubah agar *field* Server menunjuk ke server ini dan segera mengirimkan *update*nya ke server terdekat. Ada peluang redundan di sini namun sistem harus memastikan bahwa *update* yang diterima adalah yang terbaru dan ada jaminan bahwa client tetap dapat berkomunikasi meskipun kurang memuaskan. Ini merupakan salah satu keuntungan yang diberikan oleh protokol UDP.

- Inisiatif client

  Di sini, *client* proaktif untuk mengetahui apakah server dimana dirinya terhubung masih dalam kondisi "*alive*". Mekanismenya adalah dengan mengirimkan suatu pesan khusus, misalnya "/@live?" ke server dan jika mendapatkan jawaban "yes" maka tidak ada yang harus dilakukan oleh *client*. Namun, jika *client* tidak mendapakan jawaban apapun maka *client* ini harus segera mencari server alternatif dan mengubah "server aktif"nya ke server alternatif tersebut. Ini menyerupai masalah *quota* sebelumnya. Mekanisme deteksi "*alive*" ini sebaiknya dilakukan secara berkala, misalnya setiap 1 menit.

  Proses ini diperlihatkan pada gambar 7.



Gambar 7. Inisiatif client memeriksa "*alive*" tidaknya server

Komunikasi antar server sebaiknya menggunakan protokol TCP karena lebih aman terhadap hadirnya penyusup. Setiap server hanya terhubung ke server lain yang paling dekat secara topologi atau *bandwitdh*. Informasi "terdekat" ini disimpan dalam Server List. Semua *update* hanya dilakukan terhadap server lain yang terdekat.

Kesimpulan

Kami telah merancang dan mengimplementasikan suatu sistem *instant messaging* dengan server tunggal dan kemudian diperbaiki dengan menghadirkan beberapa server terdistribusi yang saling berkoordinasi untuk meningkatkan *fault-tolerance* dan *availability*. Dua aplikasi sederhana tersebut telah mampu menjawab bagaimana proses pembangunan aplikasi jaringan komputer yang memanfaatkan layer transport dan network pada model referensi OSI. Sentuhan langsung dengan level sistem memungkinkan kita mengetahui secara lebih jelas dan lengkap bagaimana proses komunikasi sesungguhnya terjadi pada suatu sistem instant messaging. Pada penelitian berikutnya, kami akan menyelesaikan

prototipe ini agar dapat digunakan secara nyata setidaknya dalam suatu jaringan kampus dan kemudian memperbaiki dan menambahkan beberapa fitur penting sehingga server-server yang terdistribusi tersebut tidak saling bergantungan.

Referensi

[1] Bjørn Elvheim, Jon Berg, Ronny Jensen (2006): DHT CHAT System**,** URL: http://www-kiv.zcu.cz/%7Eledvina/DHT/DHT-CHAT.pdf, diakses 11 Januari 2014

[2] Wikipedia (2014): Instant Messanging. URL: https://en.wikipedia.org/wiki/Instant_messaging , diakses 12 Februari 2014

[3] Introduction to Instant Messaging Software. URL: https://wikis.oracle.com/display /CommSuite/Introduction+to+Instant+ Messaging+Software, diakses 12 Maret 2014

[4] Henrik Lundgren, Richard Gold, Erik Nordstr¨om, Mattias Wiggberg (2003): A Distributed Instant Messaging Architecture based on the Pastry PeerToPeer Routing Substrate. First Swedish National Computer Networking Workshop. *SNCNW2003*. 8-10 September. URL: http://winternet.sics.se/workshops/sncnw2003/proceedings/18T-sncnw.pdf, diakses 12 Maret 2014

[5] Ha Quoc Trung (2012): New Approach To Develop The Messenger Application: From Clientserver Design To P2p Implementation, Computer Science & Information Technology (CS & IT), URL: http://www.techrepublic.com/resource-library/whitepapers/new-approach-to-develop-the-messenger-application-from-client-server-design-to-p2p-implementation/, diakses 10 Februari 2014

[6] Linan Zheng (2005): ACS Seminar – Instant Messaging: architectures and concepts. URL: http://www.ldelgado.es/seguridad/crypt4you/curso_comunicaciones_digitales/documentacion/im_architectures_and_concepts.pdf, diakses 09 Januari 2014

[7] Christian Cadruvi (2014): Extensions to a Peer-to-Peer Instant Messenger, Bachelor Thesis, ETH Zurich, URL: ftp://ftp.tik.ee.ethz.ch/pub/students/2014-FS/BA-2014-06.pdf, diakses 10 Januari 2014

[8] M.L Liu (2004): Distributed Computing Principles and Applications. Addison Wesley.

[9] Andrew Tanenbaum (2003): Computer Network, 4th Edition. Prentice Hall

[10] Jan Graba (2007): Introduction to Network Programming with Java. Springer.