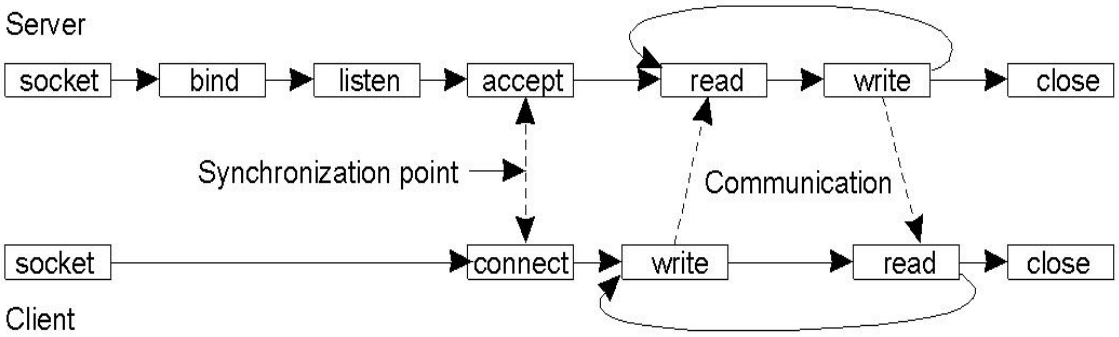
جلسه دوازدهم

ارتباط ناپایدار مبتنی بر پیام (Message Oriented Transient Com)

به عنوان یکی از ابزارهای معروفی که برای سیستم های مبتنی بر ارسال پیغام مورد استفاده قرار می گیرد، مفهوم Socket است. Socket در واقع یک communication endpoint است. یعنی یک نقطه ارتباطی نهایی است که می توان داده ای را از آن خواند یا به آن نوشت. برای استفاده از مفهوم Socket در واقع نیاز به مجموعه ای از اعمال و Primitive هاست که باید تعریف شده و توسط سیستم قابل استفاده باشد. لذا در یک سیستم مبتنی بر TCPIP معروف ترین Primitive ها یا عملیات پایه ای مربوط به Socket را می توانیم بررسی کنیم. یکی از این Primitive ها متد سوکت است که با فراخوانی این متد یا Primitive یک communication endpoint یا یک نقطه پایانی ارتباطی در داخل برنامه ای که در حال اجراست ایجاد می شود. با استفاده از Primitive ی به نام Bind در واقع یک آدرس محلی به این سوکت تخصیص داده می شود. تخصیص آدرس محلی به این منظور است که امکان نوشتن و خواندن به سوکت وجود داشته باشد. در واقع وقتی که به یک سوکت یا یک پورت آدرس محلی تخصیص داده می شود خواندن و نوشتن به آن پورت در واقع مشابه خواندن و نوشتن به یک آدرس از حافظه است و بنابراین عملیات به سادگی انجام خواهد شد.

دستور پایه ای دیگر دستور گوش دادن (Listen).می‌باشد در واقع دستور گوش دادن به این معنی به کار می رود که اعلام علاقمندی یک سرور را برای برقراری ارتباط با کلاینت ها اعلام کند. Primitive دیگر، Accept است که یک فراخواننده را تا وقتی که ارتباط مربوطه برقرار شود Block می کند. دستور Connect در واقع یک Connection یا یک ارتباط برقرار می کند و دو دستور Send و Receive نیز داده هایی را ارسال و یا دریافت می کنند و در نهایت دستور پایه ای Close باعث حذف ارتباط بین سرور و کلاینتی می شود که از تکنیک سوکت استفاده کرده اند. برای بررسی دقیق تر مفهوم فوق به شکل زیر دقت نمایید.



در این شکل دو ماشین سرور و کلاینت وجود دارند که می خواهند از Primitiveهای مربوط به سوکت برای برقراری ارتباط استفاده کنند. این ارتباط Connection oriented یا یک ارتباط مبتنی بر اتصال است. همانطور که می بینید با توجه به توصیف مربوط به Primitiveها در سطح سرور، ابتدا سوکت ایجاد می شود، سپس عملیات تخصیص آدرس(binding) انجام می شود و سپس سرور اعلام آمادگی برای برقراری ارتباط می کند(Listen) و با دستور Accept منتظر خواهد ماند تا وقتی که یک ارتباط برقرار شود. از سمت مقابل کلاینت سوکت را ایجاد می کند و با دستور Connect ارتباط را با سرور برقرار می کند لذا زمانی که دستور Connect صادر می شود، نوع عملکرد دستور Accept باعث ایجاد همگامی بین این دو فرآیند خواهد شد.

حال مجموعه ای از دستورات Write و Read مختلف در سمت سرور و کلاینت وجود دارد. اگر کلاینت Write نماید سرور می تواند Read کند و اگر سرور Write کند کلاینت Read می‌کند. در واقع هر کس که بنویسد دیگری می خواند و این سیکل آنقدر ادامه پیدا می کند تا انتقال به پایان رسیده و دستور Close صادر شود تا ارتباط بین این دو ماشین (Process) از بین برود.

ارتباط ناپایدار مبتنی بر پیام

نکته ای که در مورد سوکت ها وجود دارد این است که سوکت ها به اندازه کافی ابزار ندارند و نحوه دسترسی به آنها به سادگی انجام نخواهد شد. به این دلیل که سطح تجدید آنها و امکاناتی که در اختیار قرار می دهند، سطح تجدید مناسبی نیست؛ چرا که پس از برقراری ارتباط، تنها عملیاتی که انجام می شود عملیات read و Write است. نکته دیگر اینکه معمولاً سوکت ها با توجه به رویه پیشنهادی برای آنها تنها

پروتکل‌های عمومی (General purpose) را استفاده می کنند و امکان اینکه بتوان برای برقراری ارتباطات با سرعت بالا از پروتکل های خاص منظوره به کمک سوکت ها استفاده کرد به سادگی میسر نیست و این امکان وجود ندارد، لذا برای رفع این مشکلات از ابزارهای دیگری نیز هراز چند گاهی استفاده می شود. یکی از ابزارها به نام واسط ارسال پیغام یا Message Passing Interface یا MPI معروف است. MPI یک استاندارد است که وابسته به سخت افزار نیست. مفهومی به نام Communication server درMPI وجود ندارد یا سطح Abstracte آن یا سطح تجدید آن به قدری بالاست که این مفهوم در آن دیده نمی شود. در MPI فرض می شود که خطاهای جدی دائمی هستند و مفهومی به نام auto recovery وجود ندارد. بنابراین با این رویکرد سعی می کند که مدیریت ارسال پیغام انجام دهد. معمولاً از MPI در داخل گروههایی از فرآیندها استفاده می شود به این شکل که هر گروه از فرآیندها یک شناسه می گیرند که در روند MPI از آنها استفاده می شود. مفهوم MPI نیز می تواند در سطح Abstracte با مجموعه ای از Primitive ها یا متدهای پایه ای مدل شود که برخی از آنها مورد بررسی قرار می‌گیرد. یکی از اولین متدهای Primitive یا متدهای پایه MPI دستور MPI_bsend است، کاری که انجام می دهد این است که یک پیغام خروجی را به یک بافر ارسالی Attach کرده و در داخل آن قرار می دهد. اگر بخواهیم بین مفاهیم قبلی و دستورات MPI ارتباطی برقرار شود می توان عنوان کرد که دستورات MPI_bsendروالی مشابه این شکل ارائه می دهد. در واقع MPI_bsend یک ارتباط transient برقرار می کند که به صورت غیر همگام است. دستور دیگر MPI_send است که یک message را ارسال می کند و منتظر می ماند تا وقتی که این پیغام به بافر محلی یا Remote buffer ارسال شود. بنابراین این دستور معادل دو شکل زیر عمل می کند. MPI_ssend دستور دیگری است که یک پیغام را ارسال می‌کند و تا وقتی که دریافت انجام شود منتظر می ماند، شکل زیر معادل دستور MPI_ssend است.

MPI_sendreceive در واقع یک message را ارسال می کند و منتظر پاسخ آن می ماند. لذا شکل زیرساختار MPI_sendrecv را نشان می دهد. علاوه بر این چهار Primitive، Primitiveهای دیگری هم هستند، مانند MPI_isend که ارجاع به یک پیغام خروجی را ارسال می کند و کار خود را ادامه می دهد و هیچ انتظاری با این دستور ایجاد نخواهد شد. نکته مهم این است که در این دستور

reference یک message خروجی پاس می شود و ارسال می شود. MPI_issend ، reference یا ارجاع یک پیغام خروجی را پاس می کند یا ارسال می کند و منتظر می ماند تا دریافت آن را بگیرد یا Acnolage مربوط به دریافت آن را بگیرد. MPI_receive دستوری است که یک پیغام را Receive می کند، دستوری است که در سطح دریافت کننده بکار می‌رود و اگر پیغامی در بافر وجود نداشته باشد استفاده از این Primitive یا دستور پایه باعث متوقف شدن خواهد شد. MPI_ireceive چک می کند که آیایک پیغام ورودی وجود دارد یا خیر؟ و اگر وجود نداشته باشد اتفاق خاصی نخواهد افتاد و عملیات بلاک اتفاق نمی افتد و دریافت کننده می تواند به کار خود ادامه دهد.

تعدادی از دستورات اصلی MPI

دستورات اصلی	معنا	شکل
MPI-bsend	پیام را به انتهای بافر ارسال محلی اضافه می‌کند.	c
MPI-send	پیام را ارسال کرده و تا کپی شدن آن در بافر محلی و یا بافر راه دور منتظر می‌ماند.	(d)or (e)
MPI-ssend	پیام را ارسال کرده و تا شروع دریافت منتظر می‌ماند.	e
MPI-sendreceive	پیام را ارسال کرده و منتظر پاسخ می‌ماند.	f
MPI-isend	یک ارجاع به پیام ارسالی می‌فرستد و به کارش ادامه می‌دهد.	-

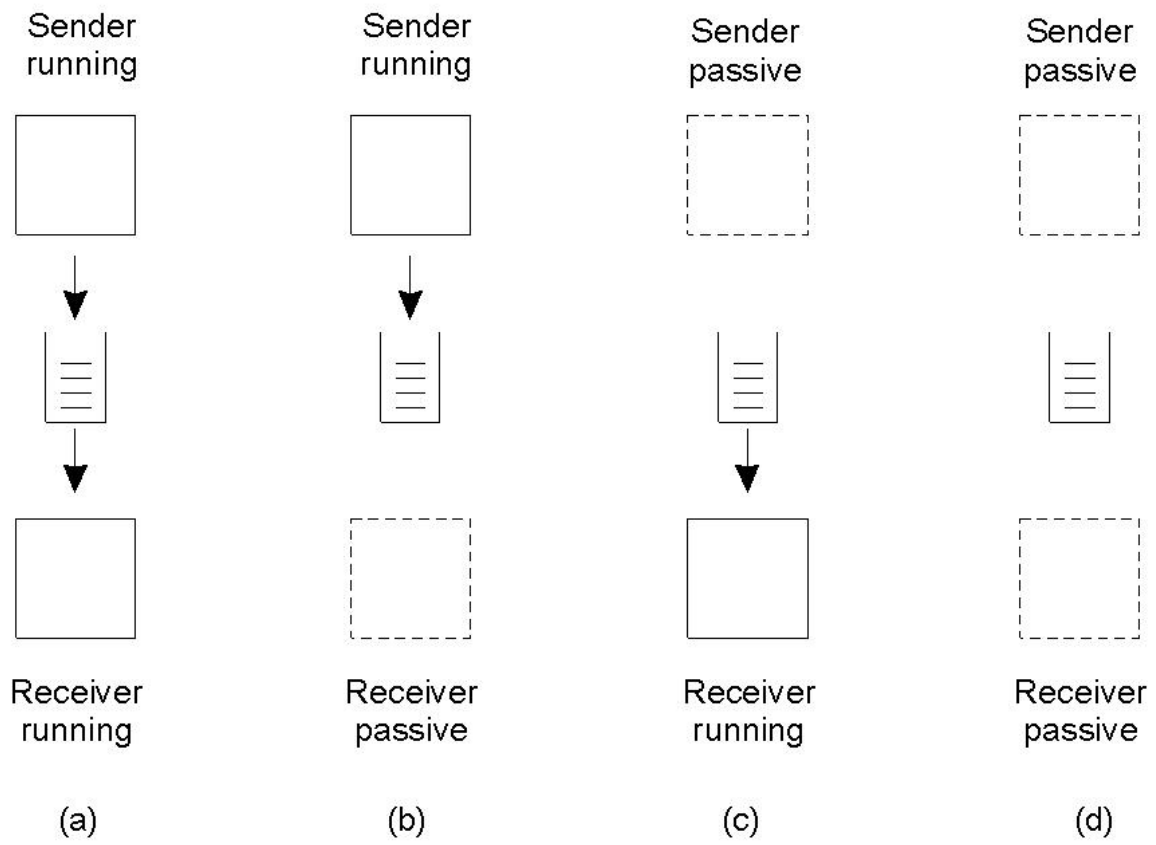
MPI-issend	یک ارجاع به پیام ارسالی می‌فرستد و تا شروع دریافت منتظر می‌ماند.	-
MPI-receive	در صورت وجود پیام آن را دریافت و در غیر این صورت بلوکه می‌شود.	-
MPI-ireceive	در صورت وجود پیام آن را دریافت و در غیر این صورت بلوکه نمی‌شود	-

ارتباط پایدار مبتنی بر پیام (Message Oriented Persistent)

تکنیک دیگری که در مدیریت مبتنی بر پیغام و از نوع persistent قابل انجام است، تکنیک مبتنی بر message-queuing system یا میان افزارهای مبتنی بر پیغام(Message oriented middlewar) ها یا MOM است. یک دسته بندی کلی از مفاهیمی که تاکنون ارائه شده است عبارتند از: سوکت Programming و Message passing interface ها یا MPIها مفاهیمی بودند که بر روی ارتباط گذرا تعریف می شدند. اما MOM مفهومی است برای ارسال پیغام به صورت ماندگار. در MOM امکان ذخیره سازی پیغام ها در فضاهای میانی وجود دارد، بدون اینکه نیاز باشد Sender یا Receiver در طول ارسال پیغام فعال باشند. در واقع این امکان فراهم می شود که از بافرها یا Storage های میانی برای ذخیره سازی پیغام ها استفاده شود و این کار باعث می شود که وابستگی به Sender و Receiver برای ذخیره سازی پیغام ها کاهش یابد. بدیهی است که آنچه که در سیستم های transient مدنظر قرار می گیرد این است که چون پیغام در بافر محلی فرستنده و گیرنده قرار می‌گیرد از بین رفتن فرستنده و گیرنده باعث آزاد شدن بافرهای مربوطه می گردد و در نتیجه پیغام نمی تواند ماندگار باشد. استفاده از بافرهای میانی و

ظرفیت ذخیره سازی (storage capacity) سایر ندهای، بین نود دریافت کننده و گیرنده این امکان را خواهد داد که با خارج شدن آنها از حالت اجرا باز هم بتوان پیغام را مدیریت کرد. MOM گارانتی می کند که پیغام مورد نظر به محض آنکه دریافت کننده فعال باشد در صف مربوط به آن قرار گیرد. این مفهوم و نحوه برخورد MOM با دریافت کننده و گیرنده در واقع اجازه یک ارتباط غیرمحکم را خواهد داد که در اصطلاح به نام ارتباطات loosely-coupled معروف اند. این امر باعث می شود که دریافت کننده و ارسال کننده بتوانند کاملاً مستقل از هم اجرا شوند بدون اینکه نیاز به دانستن وضعیت یکدیگر داشته باشند.

برای بررسی مفهوم loosely-coupled ، چهار حالت مختلف بررسی خواهند شد. در حالت ساده هم فرآیند فرستنده (Sender) و هم گیرنده در حال اجرا است. اما بافر در یک ند میانی قرار گرفته است و پیغام مربوطه در آن بافر شده است. در حالت B در هنگام ارسال ممکن است Receiver در حال اجرا نباشد. در حالت C هنگام دریافت لزوماً Sender در حالت اجرا نیست و در حالت D می بینید که پیغامی وجود دارد اما sender و Receiver هیچکدام در حال اجرا نیستند، بنابراین پیغام باقی می‌ماند تا Receiver اجرایی شود و سپس در صف مربوط به آن قرار گیرد.

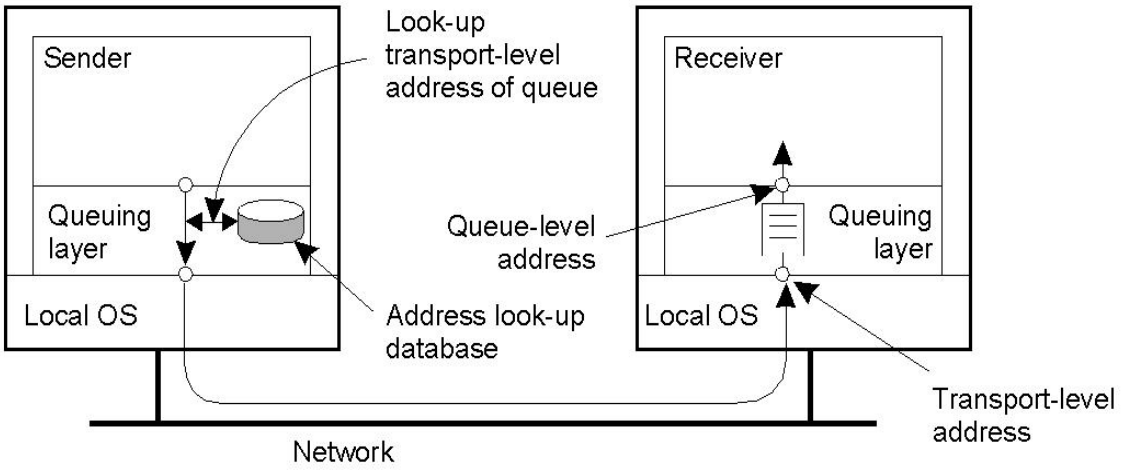


در MOM نیز مانند دو تکنیک قبلی Primitiveها یا دستورات پایه ای تعریف می شوند که با استفاده از آنها می‌توان مدیریت پیغام را در صف های مربوط به پیغام انجام داد. مهمترین این Primitiveها، Primitive put است که یک پیغام را به یک صف مشخص اضافه می کند. دیگری Primitive get است که تا وقتی که پیغام مورد نظر در صف مورد نظر قرار نگرفته باشد بلاک می شود و سپس پیغام را از صف برداشته و آن را می خواند. از دیگر Primitiveها می توان به Primitive poll اشاره کرد که در واقع یک صف را برای دریافت پیغام بررسی می کند و اولین پیغامی که در صف قرار گرفته است برمی‌دارد، اما اگر پیغامی وجود نداشت فرایندی که Primitive poll را فراخوانی کرده است بلاک نخواهد شد. Primitive دیگر Primitive notify است که یک مدیریت کننده (handler)را نصب می کند، تا وقتی که پیغامی در یک صف مشخص قرار گرفت این handler فعال شده و پیغام را پردازش کند. به نوعی می‌توان Primitive notify را مشابه مدیریت وقفه ها در سیستم های عامل محلی دانست، چرا که در مورد وقفه ها نیز معمولاً این Trapt handler فعال می شود. در اینجا نیز با استفاده از notify،

message handlerنصب می‌شود که با ورود آن message، handler مربوطه فعال و آن را پردازش می کند. در MOM امکان استفاده از توابع Call back نیز برای پردازش پیغام ها وجود دارد. در واقع توابع Call back همان message handler ها هستند که به نوعی توسط notify ثبت خواهند شد.

مفهوم یک سیستم مبتنی بر ارسال پیغام

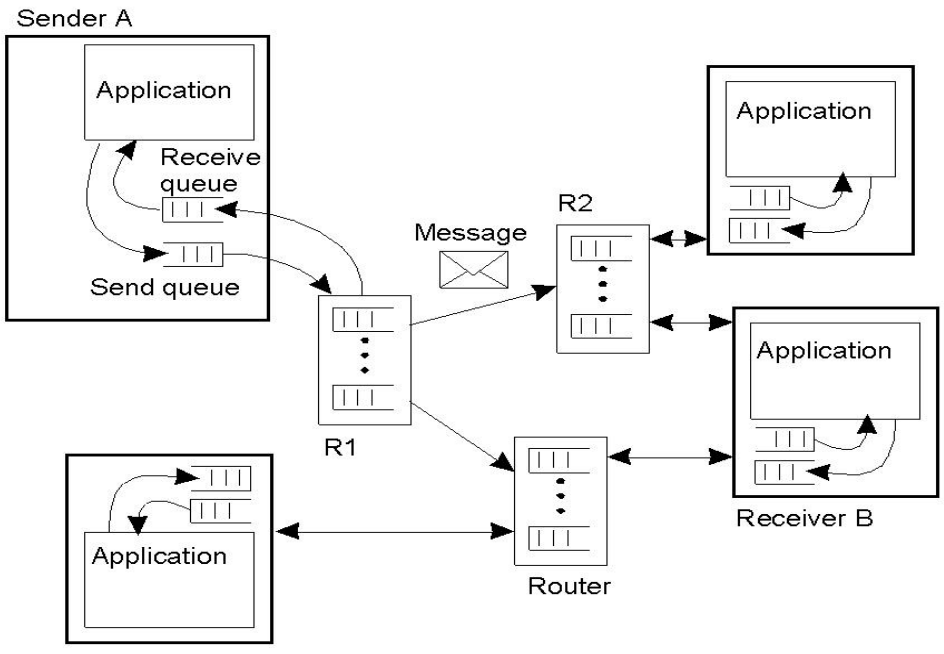
برای اینکه مفهوم یک سیستم مبتنی بر ارسال پیغام را بهتر درک کنید در اینجا سعی می شود معماری عمومی این سیستم را مورد بررسی قرار دهیم. با توجه به شکل، برای یک سیستم مبتنی برپیغام در سطح فرستنده یک لایه صف وجود دارد که این صف شامل یک پایگاه داده آدرس هاست که از این پایگاه داده برای پیدا کردن آدرس هایی که Sender می خواهد پیغامی برای آنها بفرستد، استفاده می شود.



بنابراین پیغام ارسالی ابتدا با توجه به این پایگاه داده ترجمه شده و مکان آن مشخص می شود و سپس از طریق سیستم عامل محلی و از طریق شبکه به سمت دریافت کننده ارسال خواهد شد. پایگاه داده ای که برای جستجو کردن(Look up) آدرس ها بکار می رود، مفهومی شبیه DNS در شبکه های اینترنتی دارد که اسامی را به آدرس‌های واقعی تبدیل می‌کند. همانطور که در این معماری عمومی(General Architecture)دیدیم برای Sender یک صف وجود داشت که در لایه صف مربوطه Look up آدرس انجام می شد. نکته ای که حائز اهمیت است این است که هر صف دارای یک مدیر صف است که به طور مستقیم با Application یا برنامه ای که قصد ارسال پیغام را دارد Interaction یا محاوره دارد

گرچه ذکر این نکته ضروری است که مدیر صف‌های خاصی وجود دارند که مانند routerها کار می کنند به این مدیران صف خاص RELAY گفته می شود. وظیفه RELAY ها این است که پیغام های ورودی را مستقیماً به یک مدیر صف دیگر Forward کنند و در واقع یک پیغام ورودی را به یک مدیر صف دیگر ارجاع دهند. استفاده از RELAY ها باعث می شود که ما بتوانیم سیستم های صفی بسازیم که این سیستم های صف قابلیت توسعه دارند. به عبارت دیگر، RELAYها مشابه gateway ها عمل می کنند. یعنی در واقع یک RELAY می تواند یک پیغام را از یک شبکه به یک شبکه دیگر ارسال نماید در واقع RELAY می تواند برای مفاهیم multicasting نیز استفاده شود. به نوعی با استفاده از این ساختار می توان یک سیستم مبتنی بر پیغام را بر بستر یک شبکه کامپیوتری طراحی و استفاده کرد.

معمولاً در این گونه سیستم ها، سرویس‌های نامگذاری عمومی وجود ندارد. معمولاً دارای توپولوژی استاتیک هستند و هر مدیر صف یک کپی از صف را برای انجام عملیات نگاشت محلی خود نیاز دارد. برای اینکه مفهوم مدیر صف بیشتر مشخص شود، General Architecture قبلی را با یک Level جزئیات بیشتر مورد بررسی قرار می‌دهیم. با توجه به شکل، فرستنده A، Application است که یک صف دریافت و یک صف خروجی دارد و R_1 یک RELAY است که در واقع پیغام ها را گرفته و این پیغام ها را به سمت یک Application و یا یک RELAY دیگر ارسال می کند.



بنابراین این RELAY ها خود می توانند حتی مستقلاً بر روی ماشین‌های مستقلی قرار گرفته باشند. نکته‌ای که در اینجا مهم است این است که RELAYها می‌توانند پیغام های مربوطه را به سمت یک router نیز بفرستند و این پدیده در شکل نشان داده شده است. نکته دیگر این است که ممکن است یک RELAY وظیفه برقراری و مدیریت صف مربوط به بیش از یک Application را بر عهده داشته باشد و این نکته در شکل نشان داده شده است.

مدیریت صف سیستم‌های جدید

نکته حائز اهمیتی که در سیستم های مبتنی بر پیغام وجود دارد این است که معمولاً مجتمع سازی Application های قدیمی و جدید در یک سیستم صف کار چندان آسانی نیست. به این دلیل که Application های جدید باید بتوانند پیغام هایی را که به سمت آنها می رود را بشناسند و تفسیر درستی از آنها داشته باشند. بنابراین برای رفع این مشکل شاید بهترین راه حل این باشد که تمام Application هایی که می خواهند بر بستر یک سیستم پیغام و مبتنی بر صف عمل کنند بر روی یک ساختار پیغام عمومی به توافق برسند و همگی این توافق را رعایت کنند اما اگر تعداد فرآیندها در یک سیستم زیاد باشد، معمولاً رسیدن به یک نقطه ای که تمامی فرآیندها بر روی یک ساختار پیغام به توافق برسند کار چندان ساده ای نیست و شاید تنها فرمتی که همگی در مورد آن توافق کنند یک رشته از بایت ها باشد که در آن هیچ ترتیب و اطلاع مشخصی نیست. برای اینکه این مشکل برای سیستم های مبتنی بر پیغام و MOM ها حل شود از مفهومی به نام شکننده پیغام(message broker) استفاده می شود. در واقع message broker ها مانند یک gateway عمل می کنند و کار آنها این است که پیغام‌های ورودی را به فرمتی تبدیل می کنند که بتوانند توسط فرآیند دریافت کننده قابل درک باشد و برای این کار که کار چندان ساده ای هم نیست در داخل هر message broker یک پایگاه داده قوانین وجود دارد که با استفاده از این قوانین پیغام ها ترجمه می شوند. در شکل نحوه و مکان قرار گرفتن message broker نشان داده شده است. در واقع می بینید که فرآیند ارسال کننده پیغام، پیغام را به message broker می فرستد. message broker پیغام را ترجمه کرده و به عنوان یک Sender جدید به سمت Receiver آن را ارسال می کند. بدیهی است که استفاده از چنین معماری باعث ایجاد یک Buttle neck در سیستم خواهد شد و حجم زیادی از

اطلاعات به سمت broker ارسال می شود که برای این کار نیز باید راه حلی اندیشید. یکی از راه حل‌های متداول برای حل این مشکل استفاده از سیستم‌های توزیع شده خواهد بود. به این معنی که در سیستم بیش از یک Message broker را ذخیره نماییم.