A COMMUNICATIONS SERVER, A METHOD, A USER DEVICE AND A PAYMENT GATEWAY

Technical Field

The invention relates generally to the field of communications. One aspect of the invention relates to a communications server apparatus for routing Payment Service Providers (PSP). Another aspect of the invention relates to a method, performed in a communications server apparatus for routing payment service providers. Another aspect of the invention relates to a communications device for communicating with a router for multiple payment service providers. Another aspect of the invention relates to a payment gateway. Another aspect of the invention relates to a method, performed in a router for multiple payment service providers.

One aspect has particular, but not exclusive, application in ride hailing with a large number of transactions. For example, where it may be necessary to use multiple Payment Service Providers.

Background

Various forms of payment gateway exist.

For example, in US20150178708, EP3662428, US20160071083, and WO2019144783 systems for selecting from multiple PSPs are disclosed. These prior approaches generally take a 1 step methodology to selecting a PSP for a given transaction. They may not be optimised for a wider range of failures.

Summary

Embodiments may be implemented as set out in the independent claims. Some optional features are defined in the dependent claims.

Implementation of the techniques disclosed herein may provide significant technical advantages. Advantages of one or more aspects may include:

reacting more effectively to a wider range of failure modes;

a rule-based routing layer to determine routing in a non-PSP failure mode;

a pre-emptive failover-based routing layer to respond to certain PSP failure modes; and/or

an adaptive failover-based routing layer to respond to further PSP failure modes.

In at least some implementations, the techniques disclosed herein may provide for using a three-step approach, which is much more powerful and can handle different kinds of failures better. The pre-emptive approach (applying the idea of circuit breaker pattern) may be particular useful for unexpected downtime due to random external / hardware failures. The adaptive approach (applying the idea of negative feedback loop) may be particular useful for unexpected downtime due to too much traffic.

In at least some implementations, the techniques disclosed herein may allow for:

- the technical solution of less hardware required for the same task for the technical problem of managing PSPs, due to minimised failed transactions and/or minimised retry transactions;
- the technical solution of improved reliability for the technical problem of PSP failure;
- the technical solution of faster execution for routing for the technical problem of failed PSPs;
- the technical solution of responding pre-emptively, and/or adaptively for the technical problem of failed PSPs;
- the technical solution of routing a payment to cleaner / green PSP based on the technical problem of reducing greenhouse gas emissions;

- the technical solution of reducing the data traffic transmitted between the server and the in-vehicle units based on minimised failed transactions;
- the technical solution of reducing the latency of data traffic transmitted between the server and the in-vehicle units based on faster selection of PSP; and/or
- the technical solution of reduced data centre energy requirements based on optimally managed PSPs.

In an exemplary implementation, the functionality of the techniques disclosed herein may be implemented in software running on a server communications apparatus, such as a cloud based geographically distributed database. The software which implements the functionality of the techniques disclosed herein may be contained in a computer program, or computer program product – which operates the database instances on each server node in the cloud. When running on, for example, a cloud server, the hardware features of the server may be used to implement the functionality described below, such as using the server network communications interface components to establish the secure communications channel for routing to multiple PSPs in an efficient fashion.

In one aspect there is provided a communications server apparatus for routing Payment Service Providers (PSP), the communications server comprising a processor and a memory, the communications server apparatus being configured, under control of the processor, to execute instructions stored in the memory, to:

determine based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for a given transaction;

determine the health of the applicable PSPs; and

select a PSP for the given transaction based on the health of the applicable PSPs and pre-determined criteria.

In another aspect there is provided a method, performed in a communications server apparatus for routing Payment Service Providers (PSP) related to a transportation service, the method comprising, under control of a processor of the communications server apparatus:

determining based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for a given transaction;

determining the health of the applicable PSPs; and

selecting a PSP for the given transaction based on the health of the applicable PSPs and pre-determined criteria.

In a further aspect there is provided a communications device for communicating with a PSP router comprising a processor and a memory, the communications device being configured, under control of the processor, to execute instructions stored in the memory to:

request the PSP router process a transaction;

if a plurality of transactions indicate the health of any of plurality of merchant PSPs is degraded, receiving a transaction confirmation from the PSP router based on a selection of a non degraded PSP from the plurality of merchant PSPs.

In a still further aspect there is provided a payment gateway or PSP router, comprising:

a communications server;

at least one driver communications device;

at least one user communications device; and

a communications network equipment operable for the communications server, the at least one driver communications device, and the at least one user communications device, to establish communication with each other therethrough;

wherein the driver communications device comprises a first processor and a first memory, the driver communications device being configured, under control of the first processor, to execute first instructions stored in the first memory to:

send data regarding a completed trip to the communications server;

and wherein the communications server comprises a second processor and a second memory, the communications server being configured, under control of the second processor, to execute second instructions stored in the second memory to:

initiate a transaction based on the completed trip data,

determine based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for the transaction,

determine the health of the applicable PSPs,

select a PSP for the transaction based on the health of the applicable PSPs and predetermined criteria, and

process the transaction using the selected PSP;

and wherein the user communications device comprises a third processor and a third memory; the user communications device being configured, under control of the third processor, to execute third instructions stored in the third memory to:

receive data from the communications server confirming the transaction has been successfully processed.

In a still further aspect there is provided a method, performed in a PSP router, the method comprising, under control of processors for one or more server instances:

determine based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for a given transaction;

determine the health of the applicable PSPs; and

either:

determine if the health of a given PSP is below a threshold and if so, temporarily prohibit transactions to that PSP, or

determine a success rate, and/or a failure rate of a given PSP, and limiting the maximum transactions or proportion of transactions to that PSP based on the success rate, and/or failure rate.

Brief Description of the Drawings

The invention will now be described, by way of example only, and with reference to the accompanying drawings in which:

Fig. 1 is a schematic block diagram illustrating an exemplary ride hailing service.

Fig. 2 is a schematic block diagram illustrating an exemplary communications server apparatus for routing PSPs related to a transportation service.

Fig. 3 is a schematic block diagram illustrating an exemplary PSP router architecture.

Fig. 4 is a flow diagram illustrating an exemplary processing flow.

Fig. 5 is a schematic illustrating an exemplary data structure.

Fig. 6 is a sequence diagram of an exemplary health status metric monitoring.

Fig. 7 is a timing diagram illustrating an exemplary method of storing transaction results in categories.

Fig. 8 is a state diagram of an exemplary pre-emptive failure mode routing of PSPs.

Fig. 9 is a graph of an exemplary adaptive failure mode routing of PSPs.

Fig. 10 is a screenshot of a rule entry screen.

Fig. 11 is a screenshot a rule summary screen.

Detailed Description

The techniques described herein are described primarily with reference to use in taxi, ride hailing, ride sharing, food delivery, online groceries, service / trade exchanges, and pet transport, but it will be appreciated that these techniques have a broader reach and can be usefully implemented in other fields where PSP routing may be required. Generally this might be the case in any e-commerce site with a significant number of transactions.

Figure 1 shows a ride hailing system 100, with a number of users each having a communications device 104, a number of drivers each having a user interface communications device 106, a server 102 (or geographically distributed servers) and communication links 108 connecting each of the components. Each user contacts

the server 102 using an app on the communications device 104. The user app may allow the user to enter their pick-up location, a destination address, a level of service and/or after ride information such as a rating. The level of service may include the number of seats of the vehicle, the style of vehicle, level of environmental impact and/or what kind of transport service. It may be also used to order food or other deliveries. Each driver contacts the server 102 using an app on the communications device 106. The driver app allows the driver to indicate their availability to take jobs, information about their vehicle, their location and/or after ride info such as a rating. The server 102 may then match users to drivers, based on: geographic location of users and drivers, maximising revenue, user or driver feedback ratings, weather, driving conditions, traffic levels / accidents, relative demand, environmental impact, and/or supply levels. This allows an efficient allocation of resources because the available fleet of drivers is optimised for the users' demand in each geographic zone.

Referring to Figure 2, a communications apparatus 100 is illustrated which may be used to implement the system of Figure 1. The communications apparatus 100 comprises the communications server 102, and it may include the user communications device 104 and the driver communications device 106. These devices are connected in the communications network 108 (for example, the Internet) through respective communications links 110, 112, 114 implementing, for example, internet communications protocols. The communications devices 104, 106 may be able to communicate through other communications networks, including mobile cellular communications networks, private data networks, fibre optic connections, laser communication, microwave communication, satellite communication, etc., but these are omitted from Figure 2 for the sake of clarity.

The communications server apparatus 102 may be a single server as illustrated schematically in Figure 2, or have the functionality performed by the server apparatus 102 distributed across multiple server components. In the example shown in Figure 2, the communications server apparatus 102 may comprise a number of

individual components including, but not limited to, one or more microprocessors 116, a memory 118 (e.g. a volatile memory such as a RAM, and/or longer term storage such as SSD (Solid State or Hard disk drives (HDD)) for the loading of executable instructions 120, the executable instructions defining the functionality the server apparatus 102 carries out under control of the microprocessor 116. The communications server apparatus 102 also comprises an input/output module 122 allowing the server to communicate over the communications network 108. User interface 124 is provided for user control and may comprise, for example, computing peripheral devices such as display monitors, computer keyboards and the like. The server apparatus 102 also comprises a database 126, the purpose of which will become readily apparent from the following discussion.

The user communications device 104 may comprise a number of individual components including, but not limited to, one or more microprocessors 128, a memory 130 (e.g. a volatile memory such as a RAM) for the loading of executable instructions 132, the executable instructions defining the functionality the user communications device 104 carries out under control of the microprocessor 128. The user communications device 104 also comprises an input/output module 134 allowing the user communications device 104 to communicate over the communications network 108. A user interface 136 is provided for user control. If the user communications device 104 is, say, a smart phone or tablet device, the user interface 136 will have a touch panel display as is prevalent in many smart phone and other handheld devices. Alternatively, if the label communications device is, say, a desktop or laptop computer, the user interface 136 may have, for example, computing peripheral devices such as display monitors, computer keyboards and the like.

The driver communications device 106 may be, for example, a smart phone or tablet device with the same or a similar hardware architecture to that of the user

communications device 104. Alternatively, the functionality may be integrated into a bespoke device such as a typical taxi job management device.

Thus, it will be appreciated that Figures 2 and 3 and the foregoing description illustrate and describe a communications server apparatus 102 comprising a microprocessor 116 and a memory 118, the communications server apparatus 102 being configured, under control of the microprocessor 116, to execute instructions 120 stored in the memory 118:

determine based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for a given transaction;

determine the health of the applicable PSPs; and

select a PSP for the given transaction based on the health of the applicable PSPs and pre-determined criteria.

Further, it will be appreciated that Figure 4 and the foregoing description illustrates and describes a method performed in a communications server apparatus 102, the method comprising, under control of a microprocessor 116 of the server apparatus 102:

determining based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for a given transaction;

determining the health of the applicable PSPs; and

selecting a PSP for the given transaction based on the health of the applicable PSPs and pre-determined criteria.

In e-commerce entities with higher numbers of transactions, even a slight improvement in user experience (such as card acceptance rate, transaction response latency, service uptime) makes a big difference in the everyday life of customers.

In terms of PSP management, each PSP offers different feature sets / pricing schemes / scope of connectivity / reliability. Unfortunately, there is no single best

PSP that outperforms others in every aspect. Thus, there may be a dilemma to choose which PSP to use.

Also, unexpected outages and scheduled maintenance do occur for every PSP. To maximize uptime for payment processing, there is a need to switch traffic from one PSP to another when such events occur. This may either be done manually or using a prior art failover method.

Thus, one example embodiment may use a fully-automated dynamic router, which intelligently selects the best PSP for every payment transaction, and efficiently move traffic away from any failed PSPs. This may include one or more advantages including:

It takes a federated approach to managing PSPs.

It is generic and transparent to upstream systems so that it can be easily adopted by, and useful to, new business lines and other companies.

It greatly improves the payment experience and helps to increase revenue.

Ability to concurrently route to multiple PSPs.

"Concurrently" in this case means the router could process multiple transactions at the same time. One transaction could only be sent to one healthy PSP. However, a concurrent system is able to send multiple transactions to their desired PSPs simultaneously.

A dynamic routing engine may therefore act as a proxy between a merchant domain and an acquirer domain.

Firstly the dynamic routing engine may provide for rules based routing. Each rule provides a set of conditions and the PSP that a transaction will be routed to when the rule is applied. When a transaction arrives from a certain merchant, the engine may evaluate various attributes of the transaction, sort all rules of that merchant

based on configured priority, and find the first rule whose conditions are all satisfied. Thereafter, the transaction would be routed to the PSP given in that rule.

Also, a dynamic routing engine may be used as an experimentation platform to evaluate new integration and compare the performances of existing PSPs. A rule could also provide multiple PSPs each with a given percentage, which together shall sum up to 100%. Then, the engine would randomly route transactions to each PSP following its percentage. For instance, one could set up a controlled experiment for two PSPs by routing 50% traffic to A and the other 50% to B to evaluate and compare A & B.

A dynamic routing engine may also provide an automatic failover mechanism. In one example it is possible to actively monitor the success rate of all connected PSPs in real time. When a significant success rate drop is detected from a PSP, admin users can be notified. Notifications may be useful where:

1. a routing rule does not have failover available, and admin users want to be notified when the only PSP fails so they can manually change the routing rules.
2. a routing rule has failover available, and admin users are not *required* to do anything. However, they may still wish to be notified of such failures in case they want to have offline discussion / contact with support team from the failing PSP to understand reasons / business impacts.

For rules which provide backup PSPs, traffic will be automatically switched to the backup ones. If the success rate comes back to previous level later, traffic could be routed back to original PSPs. This whole process may be performed without the need of any human intervention.

For each configured merchant, admin users can define a set of routing rules using a user interface (UI).

Figure 10 shows part of the admin UI where an admin user can create a new routing rule. In the first section, the admin user is asked to define a set of routing criteria for the new rule. The routing criteria could be either based on certain parameters of the card used in the transaction or certain parameters of the transaction itself. These criteria would determine the scope of this rule, i.e., under what situation this rule would be applicable. For example, the rule in the screenshot is only applicable when a Visa debit card 1002 issued in either Singapore or Malaysia 1004 is used to make payments, and the currency of this transaction is either Singapore dollar, Malaysia ringgit or U.S. dollar, 1006 and the route indicator sent by upstream business is either GrabTaxi or GrabFood 1008 (in Grab's use case, this indicates the payment is for a ride-hailing service or a food delivery order).

In the second section, the admin user is asked to decide the behaviour of this rule when it is applied. A routing rule could perform two actions on a transaction which matches all its routing criteria (as specified in the first section): route or reject. When the reject action is selected, the payment gateway will not forward this transaction to any PSP and will directly reject it instead. When the route action is selected, the admin user will then be required to provide a list of PSPs, each with an assigned percentage 1010. The sum of assigned percentages to all PSPs must be 100%. Then, the payment gateway would route a transaction to one of those PSPs.

Figure 11 shows the live rules listed in the order of their priority (i.e., the rule on the top of the table has the highest priority, while the one on the bottom has the lowest priority). The admin user can drag-and-drop to change the ordering of these rules (thus changing their priorities).

In Figure 11, the merchant has 3 routing rules. The 1st rule has the highest priority and is applicable for Visa card transactions. For transactions routed via the 1st rule, 40% of them will go to PSP 1, 50% of them will go to PSP 2, 10% of them will go to PSP 3. The 2nd rule is applicable to American Express credit card transactions. These

transactions are supposed to route to either PSP 1, PSP 3, or PSP 4. However, as PSP 4 is currently under a failure, it will be reassigned by the failover-based routing layer. The last rule is applicable for any transaction, and thus acts as a *"catch-all"* rule (i.e., the last rule applies for all transactions which the first two rules do not apply). This last rule has only specified one PSP, which is PSP 2. No failover-based routing is available since there is no other possible option even if PSP 2 fails.

In practical terms, the system may be implemented by an agent running as a service on one or more virtual machines "VM" within the cloud infrastructure, together with a managing proxy server. This payment router could be deployed on multiple virtual machines (VM) to distribute loads between the VMs so that the router could handle a large number of payment transactions. Here, each VM shall contain the same program and be able to independently provide a routing suggestion without interacting with other machines. These machines could be put behind a load balancer / reverse proxy which randomly forwards each incoming request to one of the machines. This architecture of multiple machines behind one load balancer could be easily achieved in most popular cloud computing providers. For example, EC2 (elastic cloud computing) instances and ELB (elastic load balancer) could be used on AWS (Amazon Web Services).

Each VM includes an operating system such as Windows Server™ 2019 or Linux e.g.: Ubutu LTS 20.04.2, with a database instance such as MySQL server community edition 5.7, hosted on AWS cloud using their RDS (relational database service) platform. However, this payment router is able to be used with any relational database (i.e., MySQL / Postgres / Microsoft SQL server / IBM DB2 / Oracle SQL server, etc).

Each VM is networked via a secure WAN within a firewall permitter. The payment router itself does not handle actual card data (i.e., full card number / expiry year & month / CVC). It only reads, writes, stores and processes general & non-sensitive

data such as issuing country / funding source / BIN, etc. Thus, the router is not required to be compliant with security standard such as PCI DSS (payment card industry data security standard). However, the router should ensure the proper authentication & authorization mechanism exists for its admin UI. To achieve adequate protection for the admin UI, the following security countermeasures could be deployed:

1. Admin users are required to perform MFA (multi-factor authentication) before performing any high-risk action (such as creating / updating / deleting a routing rule). The MFA here could be implemented in various ways, such as SMS, authenticator apps or hardware tokens.
2. Additionally, the admin UI could be protected under an VPN (virtual private network) so that only devices within enterprise / internal network could access the interface.

The system payment gateway may include various applications and running services including a web server.

The user interface 124 may be provided by an administrator logging into the routing engine via a web page accessing the proxy web server. Alternatively, a command line interface may be provided using PuTTY or similar.

The system 100 may therefore collectively be implemented in the secure WAN, the VMs within the WAN, the databases and software running on each VM, and the system payment gateway and any software running on the system payment gateway. Alternatively, the system 100 may be a single device such as a server running a database, or a user device hailing a ride utilising a transaction routed by the payment gateway. The system may be part of a larger transportation, ride sharing or ride hailing system, which allocates allocated a PSP to a transaction based on rules and/or PSP health.

The database 126 described above may include records of each user, of each driver, financial records, lat/lon of driver / vehicle geographic location, current state, vehicle information and/or type of vehicle. For example, if the vehicle is a hybrid, how many passengers it can carry, what is its maximum range, what is its CO_2 emissions on a particular route etc. It may include records of each PSP including the data fields shown in Figure 5.

Dynamic routing engine

With the addition of a routing engine 308, as shown in Figure 3, the payment gateway 304 may connect to all integrated PSPs 306 concurrently. For each incoming transaction from a business line 302, the payment gateway 304 would query the routing engine 308 to decide which PSP 306 this transaction would be routed to.

The routing engine 308 may include the following four layers:

1. Frontend admin UI layer: could be implemented using the standard HTML + CSS + JavaScript. These static files could be hosted on CDN (content delivery network) so that they could be delivered to users on different geographical locations. Some popular CDN services such as Cloudflare or CloudFront could be used.
2. Backend routing logic layer: could be implemented using any modern programming language, such as Java, Python or Go. The implementation needs to be a program which can accept HTTP requests. It also needs to talk to either the caching layer or the database layer to load all the routing rules. This program needs to be hosted on a server that runs any modern operation system such as Linux. The server needs to run the program on a certain port (such as 8080) so that the program could listen to incoming HTTP traffic.
3. Caching layer: this layer is optional, and it becomes necessary when the amount of transaction becomes large. Its main purpose is to reduce the number of reads from the database and improves the response time of the router. It could be implemented either as an embedded module in the backend layer or as a separate component. When embedded inside the backend layer, some in-

memory data structure such as hash map or LRU (least recently used) / LFU (least frequently used) cache could be used. For example, a hash map could use merchant ID as the key, and the list of rules under that merchant as the value. If there are multiple payment transactions from the same merchant, the backend layer could simply load from this hash map without looking up from database. When implemented as a standalone component, caching solutions such as Memcached or Redis could be used.

4. Database layer: could be implemented using any relational database, such as MySQL, Postgres or Microsoft SQL server. The database is used to store the routing rules of each merchant, as well as the ordering / priority of these rules.

As shown in Figure 6, standard HTTP requests and responses could be used for the payment gateway to get routing suggestion from the payment router. Alternatively, any other common remote procedure call (RPC) protocol such as gRPC or Apache Thrift could also be considered as the information could be carried in the similar fashion. Below, the communication between payment gateway 304 and routing engine 308 using HTTP protocol is explained as an example.

When the payment gateway 304 receives a request to process a payment transaction from an upstream business, it sends a HTTP request to the payment routing engine 308 to ask for routing instructions. In the body of this HTTP request, the required information will be delivered in JSON format. A sample request body is attached below:


```
{
  “txID”: “2f263d80636147dbaf58b1be12689e0f”,
  “txType”: “AUTHORISE”,
  “txInfo”: {
    “scheme”: “Visa”,
    “fundingSource”: “DEBIT”,
```

"issuingCountry": "SG",
"bin": "123456",
"currency": "SGD",
"amount": 1000,
"routeIndicator": "GrabFood",
"payerReference": "GrabUser_12345678",
"3dSecure": true
}
}

The example above is an authorise transaction, in which:

- A Visa debit card is used, and the card is issued in Singapore with a BIN (bank identification number) 123456.
- The transaction has an amount of 100 Singapore dollars.
- The transaction has a route indicator "GrabFood", which implies it is sent from the food delivery upstream business.
- The transaction has a payer reference "GrabUser_12345678", which implies the food delivery order is created by a user of the Grab app with 12345678 as its user ID.
- This transaction requires a 3D-secure (three-domain security) process as an additional verification step.

In the above request body, the first field is the transaction's globally unique ID. This ID may not be directly useful for deciding the router, but it could be helpful for tracing and/or debugging purpose later. The second field is the transaction type. The third field is a nested field which contains various parameters of the transaction. The content inside the third field could depend on the second field, as certain types of transactions may not have certain parameters.

After receiving this HTTP request, the payment router will make a routing decision based on its internal logic (rule-based routing + failover-based routing + final PSP selection). Eventually, the router will return an HTTP response to the payment gateway, whose response in JSON format carries the information required. A sample response body is shown below:


```
{
  “txID”: “2f263d80636147dbaf58b1be12689e0f”,
  “action”: “route”,
  “psp”: “PSP 1”
}
```


In the above example, the payment router returns the transaction ID to the payment gateway along with the routing instructions. The routing instructions could be one of the two possible options, either route or reject. When the decision is to route, the response body will also include an additional field to indicate which PSP is selected. The example above shows the case where the payment router decides to route the given transaction to PSP 1.

1. Rule-based routing layer:

Given a payment transaction from upstream business 302 as input, the routing engine 308 may select one of the configured downstream PSP 306 as output. This selection may be done as shown in the method 400 of Figure 4.

a. Router loads the list of pre-defined rules for this upstream business 406;
b. Router sorts these rules according to the pre-defined priority 408;
c. Router finds the first rule (based on sorting in last step) that could be applied to this current transaction 410;
   (note 1: each rule has a set of criteria that determines its applicable scope. For example, a rule can declare it is only applicable to transactions that use

Visa cards and currency in SGD. Then, non-Visa or non-SGD transactions would not be applicable to this rule.)

(note 2: a "catch-all" rule can be put at the very end. In case all previous transactions are not applicable to a specific transaction, this rule would give the "default" PSP as last resort.)

(note 3: Prior art approaches typically do not implement a rule-based layer. However, this layer may be useful in some applications because not all PSPs shall be treated "equally". For example, if PSP_X can only handle transactions from a certain country, then it cannot be used for other countries even if other PSPs have failed and PSP_X is the only one left.)

d. Router retrieves the list of allowed PSPs for this rule 412.
   (note: each rule is configured with a list of allowed PSPs, each allowed PSP has a percentage. For example, a rule could declare [PSP_A 60%, PSP_B 40%]. The percentages must add up to 100%.)

2. Failover-based routing layer:

a. Given the list of allowed PSPs with their percentages from step #1, step #2 adjusts the assigned percentage based on the health of these PSPs 414.
b. If all allowed PSPs are healthy, no adjustment would be done and proceed directly to step #3.
c. Otherwise, the list of allowed PSPs are divided into 2 groups: 1) those who are healthy; 2) those who are unhealthy.
d. The percentages assigned to PSPs in group #2 are (either fully or partially) re-assigned to those in group #1 416.
   <u>If using the preemptive approach:</u> a clear distinction between healthy and unhealthy is made. Once PSP_X is marked as unhealthy, let's say it was previously assigned 60%, then the whole 60% will be assigned to those in group #1.

<u>If using the adaptive approach:</u> healthy to unhealthy is a gradual change. The diagram on page 7 tells what portion of PSP_X's originally assigned percentage will be re-assigned.

e. The re-assignment has multiple possible methods: even distribution, to the first healthy PSP, to the last healthy PSP, etc

There is no single best method for reassignment. Rather, it should be considered on a case-by-case basis. The admin user can decide which reassignment method to adopt.

For example, if there are 3 PSPs available in the form of [PSP_A 60%, PSP_B 20%, PSP_C 20%] and PSP_B is failing.

- *Even distribution* may be used when there is no preference between PSP_A and PSP_C. When this approach is used, the percentage originally assigned to the failure PSP will be split and distributed evenly among the others. In the above example, the 20% assigned to PSP_B would be split into 2 halves (10% + 10%). One half will be given to PSP_A, the other half will be given to PSP_C. Then, the adjusted percentages would become [PSP_A 70%, PSP_C 30%].

- *Proportional distribution* could be used when we want to preserve the proportional relational between PSP_A and PSP_C. This could be used during product rollout phase for certain controller experiments. In the above example, the 20% assigned to PSP_B would be split in a way so that the proportional relationship between PSP_A and PSP_C (i.e., 3:1) remains unchanged. Thus, the adjusted percentages would become [PSP_A 75%, PSP_C 25%] so that it is still 3:1.

- *First healthy PSP* shall be used when the list of PSPs is sorted according to their preference levels (i.e., PSP_A is more preferrable than PSP_B, PSP_B is

more preferrable than PSP_C). The 20% assigned to PSP_B shall now be assigned to PSP_A. Then, the adjusted percentages would become [PSP_A 80%, PSP_C 20%].

- Similarly, *last healthy PSP* shall be used when sorted according to the reverse order of preference levels (PSP_A < PSP_B < PSP_C). The 20% assigned to PSP_B shall now be assigned to PSP_C. Then, the adjusted percentages would become [PSP_A 60%, PSP_C 40%].

Our router provides a generic and extensible interface so that it is easy to add new reassignment methods. New assignment methods could be added depending on the use case if it better suits the intention when the admin use creates the routing rule.

3. **Final PSP selection:**
   a. Given the list of allowed PSPs with (adjusted) percentages, step #3 selects one of them 418. A larger percentage means a higher possibility to be selected.

      Let' say there are 3 PSPs available in the form of [PSP_A 60%, PSP_B 20%, PSP_C 20%]. The actual selection process is as follows:
      - Generate a random number *r* in the range of [0, 1].
      - If r is in the range of [0, 0.6], then select PSP_A; if in the range of [0.6, 0.8], then select PSP_B; otherwise, select PSP_C.

      By the law of large numbers, when this process happens repeatedly for a lot of times, the frequency for each PSP would be equal to its assigned percentage.

(note: *[PSP_A 60%, PSP_B 40%] means PSP_A has a 60% chance while PSP_B has a 40% chance.*)

Rules Based Routing

There are two tables in the database, the routing_rule table and the routing_sequence table whose schema is shown in Figure 5.

Each row in the routing_rule table 500 describes a rule with its conditions and configured PSPs (with corresponding percentages). Each row in the routing_sequence table 502 contains an ordered list of rule_ids to define the priority of rules under each merchant.

To route an incoming transaction, the engine first loads the routing_sequence record according to the merchant_id. Following the order specified in the routing_sequence record, it then loads each rule_id and checks whether the transaction satisfies the rule's condition. If so, it will route the transaction to (one of) the defined PSP(s) in that rule.

As mentioned above, routing rules are a list of pre-configured rules for every upstream business. They are primarily used in step #1 of the route selection process.

Each rule needs to declare two sets of parameters: 1) a list of routing criteria (to decide when this rule is applicable); and 2) a list of allowed PSPs with their assigned percentages.

Each rule is expressed in the format of <facet, operator, value>. Possible facets include: amount, currency, card scheme, payer identifier, funding source, BIN, issuing country, issuing bank, transaction time, etc. Possible operators diff for each facet, but could be >, >=, <, <=, =, <>, IN.

If the list of routing criteria is empty, it means this rule is applicable to any transaction. Thus, it becomes a “catch-all” rule.

If the list of allowed PSPs is empty, it means transactions applicable to this rule will be rejected.

An example rule:

Criteria: currency IN (SGD, MYR), cardScheme = Visa, amount > 100.

Allowed PSPs: [PSP_A: 60%, PSP_B: 40%]

This rule means: for all transactions on Visa cards with an amount larger than 100SGD or 100MYR, 60% will go to PSP_A while 40% will go to PSP_B.

It will be stored in the “providers” column of the “routing_rule” table. For instance, the above example rule would be stored in the routing_rule table in the following way:

rule_id	merchant_id	conditions	providers	is_enabled

19658901	12	{
		[
		“facet”: “currency”,
		“operator”: “in”,
		“value”: “SGD,MYR”
		],
		[
		“facet”: “scheme”,
		“operator”: “equal”,
		“value”: “Visa”
		],
		[
		“facet”: “amount”,
		“operator”: “larger”,
		“value”: “100”
		]
		}
			{
			“PSP_A”: 60,
			“PSP_B”: 40
			}
				true

(created_at and updated_at columns are ignored here as they are simple timestamps for tracing purpose)

Using the rules-based routing may avoid certain kinds of “failures” in the domain of PSPs. For example, routine maintenance: one PSP may be known to have downtime every Monday 3AM because of scheduled maintenance.

Such failures are usually known facts before hand, and thus failover could easily be achieved using our step #1 (by configuring a routing rule to provide a blackout time for the PSP under maintenance).

As mentioned earlier, a rule has two possible actions: route or reject. For blackout scenarios, the rule will be “reject”.

Prior art methods, particularly machine learning would probably be ineffective at this. Machine learning models need a large amount of data and computational resources to learn patterns over a relatively long period of time. Machine learning may not be suited to small or medium sized enterprises who do not have big data. A rule-based method may save a lot in terms of time and resources. Also given a known fact, a rule-based method could guarantee 100% accuracy. Although machine learning may become more and more accurate over time, it is an iterative process (i.e., from 90% accurate to 99% accurate to 99.9% accurate).

Real-time Metrics Monitoring

To detect when a PSP starts to fail as well as when the failure stops, its success rate may be monitored in real time. Effectively, the routing engine 308 needs to know the results of all transactions and keep track of the successful ones and the failure ones. This feedback loop is highlighted in the sequence diagram in Figure 6.

The payment gateway 304 will inform the routing engine 308 about the transaction results in an asynchronous manner via a message stream.

A message stream may be created between the payment gateway 304 and the routring engine 308, where the payment gateway 304 acts as the publisher and the routing engine 308 acts as the subscriber. Possible choices for the message stream include Apache Kafka, Apache Flink, etc.

The message stream here does not have to be a FIFO (first in first out) queue. Rather, the messages inside could be loosely ordered if, depending on the application requirements, there are no strict requirements on the ordering of messages.

After a transaction has been sent to an PSP for processing (no matter success of fail), the payment gateway 304 would publish a message to the message stream. Due to the nature of the stream, the payment gateway 304 does not need acknowledgement from the subscriber (i.e., the routing engine 308). The routing engine 308 would then listen to the message stream and read whenever there is a new message in the message stream. The format of the messages in the message stream is as follows:


```
{
   “txID”: “2f263d80636147dbaf58b1be12689e0f”,
   “txTime”: “2021-05-16T07:07:03+00:00”,
   “txStatus”: “success”,
   “psp”: “PSP_A”,
   “txInfo”: {
      “scheme”: “Visa”,
      …
   }
}
```

In the above example message, the first field is the transaction's unique ID for tracing purpose. The second field is a timestamp which represents when this transaction was processed. The third field describes whether the transaction succeeds or fails. The fourth field tells which PSP was used. The last field includes all parameters of this transaction.

This may have one or more of the following benefits:

Reduce response time for upstream business lines, as the payment gateway could respond immediately without waiting for routing engine to process the metrics; and/or

Decouple metrics processing from online transaction processing. Even if the metrics collector fails unexpectedly, payment processing would still carry on without any interruption.

With this configuration, the routing engine 308 may recognise the result of every transaction.

In certain applications it may be useful to determine the average success rate of a given PSP over the past n minutes. This might be termed a rolling average. An evaluation window may be defined as "the past n minutes", which is configurable for every PSP in the management interface as shown below:

PSP	**Evaluation Window**
PSP_A	5m
PSP_B	60s
PSP_C	3m

One implementation is calculating the success and failure rates using a database table, and store every transaction result with its timestamp as a row. To calculate the success rate, the table could be queried to count the number of successful (and

failure) transactions whose timestamp falls into the evaluation window, then divide the counts by the total transactions in that window.

However, this approach has two problems:

- It wastes a lot of storage space (and thus incurs higher cost). Ideally, we only want a single counter to store the number of transactions. However, this approach stores every transaction and its timestamp, which may be too detailed. It increases the space complexity from the ideal O(1) to O(n).
- It is not efficient. When the transaction volume is huge, this approach has to count a large number of rows which would be very slow.

Another implementation is to use a sliding-window bucket-based approach, or rolling average. This method maintains a mapping from time window to buckets (i.e., each bucket contains two counters, one for successful transactions and the other for failure ones). When a transaction result is received, we don't store it individually. Rather, we find out which bucket it belongs to and increments the counter.

In Figure 7, an example of a sliding window bucket approach is shown where the evaluation window is 5 minutes and bucket size 1 minute. Let's assume the current timestamp is 19:50:20. We can sum up the buckets A - E (i.e., from 19:45:00 to 19:50:00) to approximate the past 5 minutes (19:45:20 to 19:50:20). Although this approach may bring inaccuracy, it is generally within an acceptable range.

Users are allowed to decide an appropriate value for bucket size on the management interface. Given the same evaluation window, a smaller bucket size would use more storage space but lead to better accuracy; a larger bucket size would be more space-efficient but result in more approximation.

The selection of bucket size is a trade-off between resources used (i.e., the amount of storage space needed) and accuracy achieved (the difference between the approximated sliding window versus the real sliding window).

An appropriate measurement for the bucket size is its relative comparison to the sliding window size rather than its absolute size. In other words, we need to measure how many buckets each sliding window would consist of. In the example, the window size is 5 minutes while bucket size is 1 minute. Thus, each sliding window consists of 5 buckets.

When each window consists of *n* buckets, the error could be at most 1/n. Thus, the accuracy level would be at least (1 – 1/n). In the example, the accuracy level would be at least 80%. For example, n may be at least 5 to achieve useful accuracy.

Automatic Failover & Recovery

The metrics collected above can be used to detect when a failure from a certain PSP has started and when it has recovered. Two approaches, are proposed: the pre-emptive approach and the adaptive approach.

The following terms are used below:

t: a number in the range of [0, 1], which is the average success rate over the evaluation window of a given PSP;

Cooldown period: the minimum amount of time that a PSP has to stay in open state;

Trial: a small number of transactions that would be routed to PSP under failure to check whether it has recovered;

Trial size: the number of transactions in a trial; and

tt: a number in the range of [0, 1], which is the average success rate within a trial of a given PSP.

Pre-emptive failure mode routing

In this approach, we follow a circuit breaker pattern and define 3 states: closed, open and half-open. The transition between these 3 states 800 are described in Figure 8.

By default, every PSP is under closed state 802 when there is no failure. The failover engine periodically checks the average success rate, and moves it to open state 804 when the success rate drops below the configured threshold 806.

The success rate of payment transactions depends on many external factors. Even when a PSP is healthy, we cannot expect it to successfully process every transaction. Thus, a reasonable threshold should be set based on past data. For example, 94% is generally a good threshold for most PSPs. For certain PSPs (usually local providers in less-developed countries where issuing banks and acquirers are generally less stable), this value could be lower such as 85%.

The threshold could be gradually fine-tuned as the system runs in production.

Under open state 804, traffic will be switched away from the PSP and the backup PSP (if configured) would be used. Or putting it differently the unhealthy PSP is temporarily blocked or inhibited. The PSP would stay in the open state 804 until the configured cooldown period 808 has passed. Then, the PSP would be moved to half-open state 810, where a trial 812 would be conducted. During the trial 812, a controlled small number of transactions would be routed back to the PSP under failure. Based on the results of these transactions, we could evaluate whether the PSP has recovered from the failure. The rationale here is: we can only determine the recovery of an incident by evaluating the success rate of the given PSP. Without letting some volume go through the PSP, we could not calculate the success rate effectively. Eventually based on whether the success rate in the trial 812 has improved above the threshold 814, the PSP would be moved back to closed state

802. If after all the trials, the success rate is below the threshold 816 the PSP would be moved back to the open state 804. Then the process repeats.

The trial size here could not be too large. This is because we are unaware whether the PSP has recovered at the point of the trial. If the PSP is still failing, the transactions within the trial would still fail. However, a too small trial size would not be representative enough as well.

For example, a value between 50 and 100 may be an effective and efficient trial size. Threshold 814 may be the same as threshold 806.

If a PSP stays failed after the retries, this process of repeating between open state and half-open state may continue.

Using the Pre-emptive failure mode routing may avoid certain kinds of “failures” in the domain of PSPs. For example, unexpected downtime due to random external / hardware failures: the server of an PSP may suddenly have a power failure and fail. There is no pattern to such failures and have no relation to the attributes of transactions.

Machine learning based PSP management might attempt to determine a predicted performance capability of each of a plurality of gateways based on one or more transaction parameters associated with the data. However, for this type of failure, machine learning would fail, as there is no correlation between hardware failures with the transaction parameters. When a PSP’s server fails, transactions routed to that PSP would fail no matter their parameters.

The pre-emptive approach may be used to identify such failures. Since such failures may not be predictable, real-time monitoring and failover as soon as real-time metrics indicate a failure may provide an improved outcome.

Adaptive failure mode routing

The adaptive approach, works without maintaining the states of the pre-emptive approach. This method does not make a clear distinction between a normal PSP and a failed PSP. Rather, it models the whole failover process as a probability function that changes with the observed success rate. At any time, we define failover rate q as the percentage of transactions that would be routed to backup PSPs. In other words, we want q = 0 when PSP is healthy and q = 1 when PSP is under outage. We can express q as a function of success rate t:

$$q = max(0, min(1, (a - t)^b + c))$$

In this probability function, a and c are just constants that determine the two inflection points, as mentioned in page 7.

If the success rate is 100%, it sticks with the originally allocated percentages.

As the success rate reduces, the percentage allocated to the failed PSP would gradually be reduced and reallocated to other healthy PSPs. This smooth & gradual reassignment process is governed by the above formula.

The value b determines how quick q would change with respect to t. When b = 1, we say q would be strictly inverse proportional to t. However, in the real world, we would want q to change faster than that. A larger b means the failover router is more sensitive to the success rate / latency of the PSP.

- The value of a, b and c needs to be determined by admin users when the system is setup.
- It is possible that each PSP reacts differently to load. Over long period of time, the admin users could fine-tune the parameters a, b and c based on each PSP's reaction to load.

An example is plotted in Figure 9 when a = 1.65, b = 12 and c = -0.02. The horizontal axis (t-axis) represents the success rate of a specific PSP; the vertical axis (q-axis) represents the failover rate for that PSP. Let's say this PSP is originally assigned 60% of the traffic and failover rate is q. Then, 60% * q will be reassigned to other PSPs, while 60% * (1 - q) will remain with this PSP.

In the above example, we notice the function has two inflection points, point X and point Y. At the right side of point X, the success rate from the PSP is high enough so we won't route any transaction to the backup PSP. Then between point X and point Y, traffic would be gradually moved to backup PSP. The rate of traffic movement becomes higher and higher as the incident becomes more serious (i.e., normally an incident has a larger impact when success rate is lower and more transactions are rejected). At the left side of point Y, we believe the state of the original PSP is too unstable to accept any traffic. Thus, all traffic would be switched to the backup one. The adaptive approach provides a smoother transition from original PSP to backup PSP. However, both approaches should behave reasonably well in most cases.

Using the adaptive failure mode routing may avoid certain kinds of "failures" in the domain of PSPs. For example, unexpected downtime due to too much traffic: a PSP is having downtime / becomes slower / has more errors because we send too high volume of transactions to it.

As indicated in Figure 9, as the success rate from a PSP goes down (or becomes slower), we reduce the amount of transactions sent to it. As the volume decreases, we expect the PSP to gradually recover. Then as the PSP's success rate goes back, the volume to that PSP can be increased.

When load-based failure occurs to one PSP, the router would switch traffic to other healthy PSPs who have not caused load-based failure yet.

If all available PSPs are having load-based failure, the router would not do any reassignment at all. Rather, it would distribute traffic based on the originally assigned percentages in the routing rules.

For planned outages, the rules-based approach may be more accurate and cost-effective. For unpredictable failures the pre-emptive approach may be a faster response and quicker recovery. For predictable failure, the adaptive approach may provide an effective way to manage the load between each PSP to avoid an increased failure rate. It is possible for the admin user to set one merchant to use pre-emptive approach, while another merchant can use the adaptive approach

It will be appreciated that the invention has been described by way of example only. Various modifications may be made to the techniques described herein without departing from the spirit and scope of the appended claims. The disclosed techniques comprise techniques which may be provided in a stand-alone manner, or in combination with one another. Therefore, features described with respect to one technique may also be presented in combination with another technique.

Claims

1. A communications server apparatus for routing Payment Service Providers (PSP), comprising:

a communications server; and

a communications network equipment operable to establish communication between the communications server, at least one driver communications device, and at least one user communications device;

wherein the driver communications device comprises a first processor and a first memory, the driver communications device being configured, under control of the first processor, to execute first instructions stored in the first memory to:

send data regarding a completed trip to the communications server;

and wherein the communications server comprises a second processor and a second memory, the communications server being configured, under control of the second processor, to execute second instructions stored in the second memory to:

initiate a transaction based on the completed trip data,

determine based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for the transaction,

determine the health of the applicable PSPs,

select a PSP for the transaction based on the health of the applicable PSPs and predetermined criteria, and

process the transaction using the selected PSP;

and wherein the user communications device comprises a third processor and a third memory; the user communications device being configured, under control of the third processor, to execute third instructions stored in the third memory to:

receive data from the communications server confirming the transaction has been successfully processed.

2. The communications server apparatus of claim 1 wherein transactions are routed to each of the stored PSPs concurrently.

3. The communications server apparatus of claim 1 or 2 wherein the plurality of pre-determined rules may include which of the plurality of stored PSPs can be used for one or more corresponding transaction types.

4. The communications server apparatus of any preceding claim wherein the plurality of pre-determined rules may further include blackout periods for one or more of the plurality of stored PSPs.

5. The communications server apparatus of any preceding claim wherein the health of the applicable PSPs is determined using a rolling average of a success rate, and/or a failure rate.

6. The communications server apparatus of claim 6 wherein the rolling average is determined using a sliding-window bucket approach.

7. The communications server apparatus of any preceding claim wherein the pre-determined criteria include determining if the health of a given PSP is below a first threshold and if so, prohibit transactions to that PSP.

8. The communications server apparatus of claim 7 wherein the pre-determined criteria further include determining if the health of the failed PSP have recovered above a second threshold and if so, allowing transactions to that PSP.

9. The communications server apparatus of any preceding claim wherein the pre-determined criteria include determining the success rate, and/or failure rate of a given PSP, and limiting the maximum transactions or proportion of transactions to that PSP based on the success rate, and/or failure rate.

10. A method, performed in a PSP router, the method comprising, under control of processors for one or more server instances:

determine based on a plurality of pre-determined rules which of a plurality of stored PSPs are applicable for a given transaction;

determine the health of the applicable PSPs; and

either:

determine if the health of a given PSP is below a threshold and if so, temporarily prohibit transactions to that PSP, or

determine a success rate, and/or a failure rate of a given PSP, and limiting the maximum transactions or proportion of transactions to that PSP based on the success rate, and/or failure rate.

11. A computer program or computer program product comprising instructions for implementing the method of claim 10.

12. A non-transitory storage medium storing instructions, which when executed by a processor, cause the processor to perform the method of claim 10.