```
pragma solidity ^0.4.4;

contract ServiceLevelAgreement {

 address provider;  //Stores the address of the provider

 address consumer;  //Stores the address of the consumer

 uint endTime;   //Stores the time when the contract will expire

 uint transactions = 0;  //Accumulates the transactions done so far

 uint providerAmount = 0;  //Amount owed to the provider

 uint consumerAmount = 0;  //Amount owed to the consumer

 function ServiceLevelAgreement() {
  // The creator of the contract is the provider
  provider = msg.sender;
 }

 modifier onlyProvider() {
  require(msg.sender == provider);
  _;
 }

 modifier onlyConsumer() {
  require(msg.sender == consumer);
  _;
 }

 event RegisteredSLA(address indexed _provider, address indexed _consumer, uint indexed _timestamp);
```

```
  event ServiceUnavailability(address indexed _provider, address indexed _consumer, uint _timestamp, uint indexed amount);


  event ThroughputPayout(address indexed _provider, address indexed _consumer, uint _timestamp, uint indexed amount);


  //Function to register and legitamize the SLA and an event is trigerred on registration
  function registerSLA(address _consumer, uint _endTime)
                    onlyProvider
  {
    consumer = _consumer;
    endTime = now + _endTime;


    RegisteredSLA(provider, consumer, now);
  }


  //Function called by the provider to store the number of transactions completed so far.
  //The amount accured by the provider too is calculated and an event is triggered
  function transaction(uint _transactions)
          onlyProvider
  {
    if(now < endTime)
    {
      transactions += _transactions;
      if(transactions >= 5){
        providerAmount += transactions/5 * 10;
        transactions -= transactions/5;
      }
    }
    else
      return;
    ThroughputPayout(provider, consumer, now, providerAmount);
```

```
}

//Function called by the consumer to report an infraction
//The amount accured by the consumer too is calculated and an event is triggered
function reportInfraction()
        onlyConsumer
{
  if(now < endTime)
  {
    consumerAmount += 10;
  }
  else
    return;
  ServiceUnavailability(provider, consumer, now, consumerAmount);
}

//Function to pay the recepient the required value
function payValue()
        payable
        returns(bool)
{
  if(now < endTime)
  {
    if(msg.sender == consumer){
      if(msg.value <= providerAmount){
        if(!provider.send(msg.value)) throw;
        providerAmount -= msg.value;
        return true;
      }
      else
        return false;
    }
```

```
    else if(msg.sender == provider){
      if(msg.value <= consumerAmount){
        if(!consumer.send(msg.value)) throw;
        consumerAmount -= msg.value;
        return true;
      }
      else
        return false;
    }
    else
      throw;
  }
  else
    throw;
 }

}
```