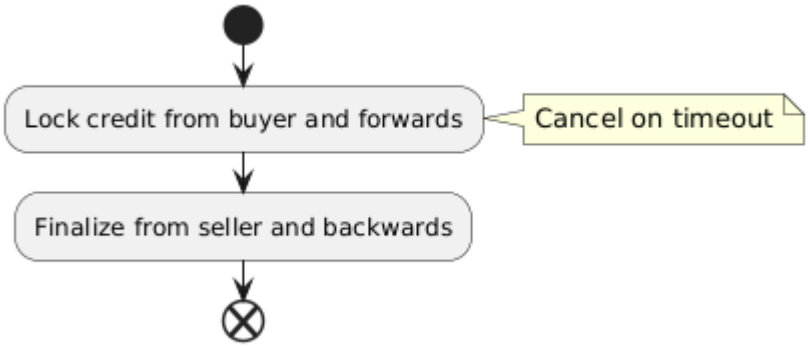
Perfect solution to Denial of Service (DoS) attacks in multi-hop payments

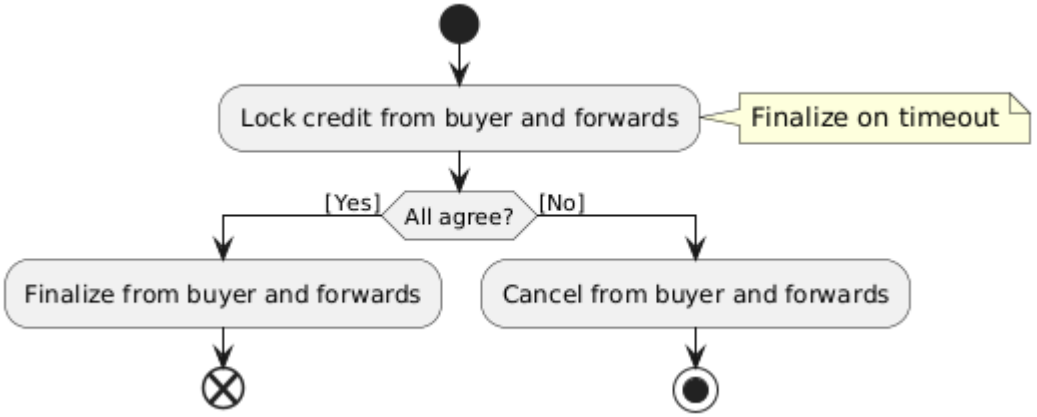
Johan Nygren, @Bipedaljoe

The basic building block to prevent Denial of Service attacks is a timeout. A timeout can be used in exactly two ways. It can cancel the payment, or it can finish the payment. Used alone, either timeout configuration will always risk punishing a node, but either timeout configuration will always have DoS vectors that have no penalty. The penalty can be reduced in size either by doing continuous timeouts (as Ryan Fugger suggested in 2006 here on Sourceforge Ripple mailing list), or, by reducing the size of the payment (as Interledger did with "stream payments"). The alternative to reduce the size of the payment itself ("stream payments") is not a good solution. The "chunked penalty" is the ideal solution, but while it is able to deter the DoS vectors that have a penalty it will tend to introduce vulnerability where there is no penalty (as the "continuous timeouts" tend to increase the overall total length until the payment has fully timed out). I.e., "chunked penalty" as a solution to the problem of too big penalty means the original solution of a timeout only works where the timeout incurs a penalty but not where it does not. The solution is to make sure every DoS vector has a penalty, and to achieve that, both timeout configurations (cancel on timeout and finish on timeout) have to be combined.

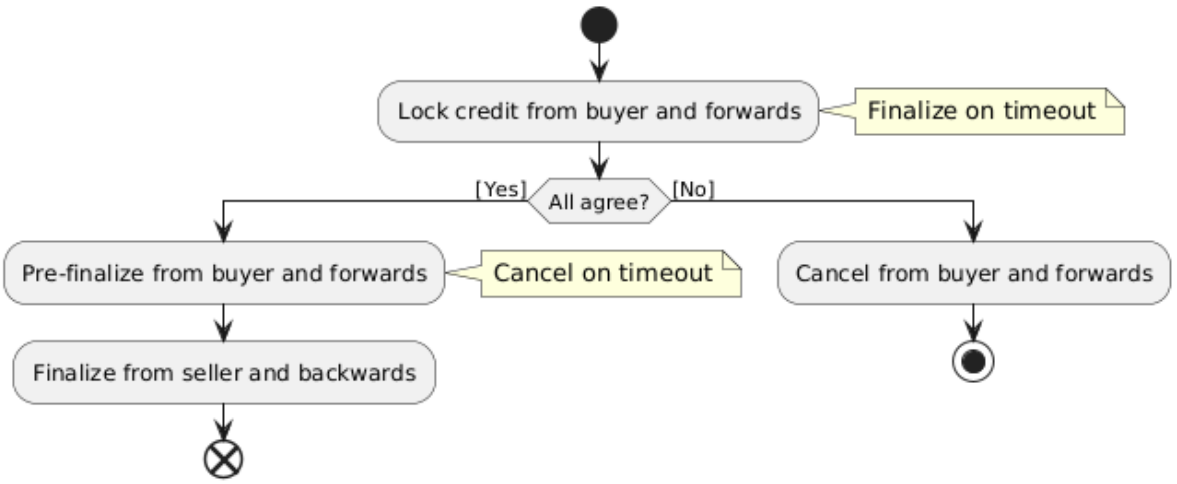
All multi-hop payment systems in the past 20 years have been built on a timeout that defaults to cancel, with a penalty risk when finishing the payment from the end recipient and backwards. Here, the "finish payment" step (what Ryan Fugger called "commit") has a penalty, but the "start payment" step (what Ryan Fugger called "commit-ready") does not. Thus, with a "chunked penalty", the "start payment" step would become vulnerable to DoS attacks (whereas the "finish payment" step was defended). (Interledger resolved this by using "chunked payments" instead of "chunked penalty" but this is not a good solution).



The other way to use a timeout is one that finishes the payment. It is equally good to the one that cancels on timeout. Both have penalty at one step (either "start" or "finish" the payment) but no penalty at the other step (for finish on timeout, the "start payment" has a penalty that enforces the original sender to cancel the payment, an action that has to be authenticated with a hash lock just like the "finish payment" step has to be in the "cancel on timeout" configuration and for similar reason).



So a timeout can be used in exactly two ways, both introduce a penalty only on one of the two steps (either "start" or "finish" the payment), but they introduce the penalty on opposite steps. To have a penalty at both steps, both timeout configurations can be simply combined.



This is a very simple idea. But since either timeout configuration on its own seems to almost be the solution, it is also easy to miss the fact that their shortcomings are exact opposites.

The combined timeout configurations penalize DoS attacks in every scenario except when the attacker controls both ends of the penalty (the sender and receiver). To also deter attacks in that scenario, fees added on top of payment (and paid out in proportion to how long payment was stuck) are also needed.

Flow chart code to render the images

```
@startuml
start
:Lock credit from buyer and forwards;
note right: Cancel on timeout
:Finalize from seller and backwards;
end@enduml
```

```
@startuml
start
:Lock credit from buyer and forwards;
note right: Finalize on timeout
if (All agree?) then ([Yes])
:Finalize from buyer and forwards;
end
else ([No])
:Cancel from buyer and forwards;
stop
endif@enduml
```

```
@startuml
start
:Lock credit from buyer and forwards;
note right: Finalize on timeout
if (All agree?) then ([Yes])
:Pre-finalize from buyer and forwards;
note right: Cancel on timeout
:Finalize from seller and backwards;
end
else ([No])
:Cancel from buyer and forwards;
stop
endif@enduml
```