Blockchain Applications

Let's now build on our understanding of bitcoin by looking at it as an *application platform*. Nowadays, many people use the term "blockchain" to refer to any application platform that shares the design principles of bitcoin. The term is often misused and applied to many things that fail to deliver the primary features that bitcoin's blockchain delivers.

In this chapter we will look at the features offered by the bitcoin blockchain, as an application platform. We will consider the application building *primitives*, which form the building blocks of any blockchain application. We will look at several important applications that use these primitives, such as payment (state) channels and routed payment channels (Lightning Network).

Introduction

The bitcoin system was designed as a decentralized currency and payment system. However, most of its functionality is derived from much lower-level constructs that can be used for much broader applications. Bitcoin wasn't built with components such as accounts, users, balances, and payments. Instead, it uses a transactional scripting language with low-level cryptographic functions, as we saw in [transactions]. Just as the higher-level concepts of accounts, balances, and payments can be derived from these basic primitives, so can many other complex applications. Thus, the bitcoin blockchain can become an application platform offering trust services to applications, such as smart contracts, far surpassing the original purpose of digital currency and payments.

Building Blocks (Primitives)

When operating correctly and over the long term, the bitcoin system offers certain guarantees, which can be used as building blocks to create applications. These include:

No Double-Spend

The most fundamental guarantee of bitcoin's decentralized consensus algorithm ensures that no UTXO can be spent twice.

Immutability

Once a transaction is recorded in the blockchain and sufficient work has been added with subsequent blocks, the transaction's data becomes immutable. Immutability is underwritten by energy, as rewriting the blockchain requires the expenditure of energy to produce Proof-of-Work. The energy required and therefore the degree of immutability increases with the amount of work committed on top of the block containing a transaction.

Neutrality

The decentralized bitcoin network propagates valid transactions regardless of the origin or content of those transactions. This means that anyone can create a valid transaction with sufficient fees and trust they will be able to transmit that transaction and have it included in the blockchain at any time.

Secure Timestamping

The consensus rules reject any block whose timestamp is too far in the past or future. This

ensures that timestamps on blocks can be trusted. The timestamp on a block implies an unspent-before guarantee for the inputs of all included transactions.

Authorization

Digital signatures, validated in a decentralized network, offer authorization guarantees. Scripts that contain a requirement for a digital signature cannot be executed without authorization by the holder of the private key implied in the script.

Auditability

All transactions are public and can be audited. All transactions and blocks can be linked back in an unbroken chain to the genesis block.

Accounting

In any transaction (except the coinbase transaction) the value of inputs is equal to the value of outputs plus fees. It is not possible to create or destroy bitcoin value in a transaction. The outputs cannot exceed the inputs.

Nonexpiration

A valid transaction does not expire. If it is valid today, it will be valid in the near future, as long as the inputs remain unspent and the consensus rules do not change.

Integrity

A bitcoin transaction signed with SIGHASH_ALL or parts of a transaction signed by another SIGHASH type cannot be modified without invalidating the signature, thus invalidating the transaction itself.

Transaction Atomicity

Bitcoin transactions are atomic. They are either valid and confirmed (mined) or not. Partial transactions cannot be mined and there is no interim state for a transaction. At any point in time a transaction is either mined, or not.

Discrete (Indivisible) Units of Value

Transaction outputs are discrete and indivisible units of value. They can either be spent or unspent, in full. They cannot be divided or partially spent.

Quorum of Control

Multisignature constraints in scripts impose a quorum of authorization, predefined in the multisignature scheme. The M-of-N requirement is enforced by the consensus rules.

Timelock/Aging

Any script clause containing a relative or absolute timelock can only be executed after its age exceeds the time specified.

Replication

The decentralized storage of the blockchain ensures that when a transaction is mined, after sufficient confirmations, it is replicated across the network and becomes durable and resilient to power loss, data loss, etc.

Forgery Protection

A transaction can only spend existing, validated outputs. It is not possible to create or counterfeit value.

Consistency

In the absence of miner partitions, blocks that are recorded in the blockchain are subject to reorganization or disagreement with exponentially decreasing likelihood, based on the depth at which they are recorded. Once deeply recorded, the computation and energy required to change makes change practically infeasible.

Recording External State

A transaction can commit a data value, via OP_RETURN, representing a state transition in an external state machine.

Predictable Issuance

Less than 21 million bitcoin will be issued, at a predictable rate.

The list of building blocks is not complete and more are added with each new feature introduced into bitcoin.

Applications from Building Blocks

The building blocks offered by bitcoin are elements of a trust platform that can be used to compose applications. Here are some examples of applications that exist today and the building blocks they use:

Proof-of-Existence (Digital Notary)

Immutability + Timestamp + Durability. A digital fingerprint can be committed with a transaction to the blockchain, proving that a document existed (Timestamp) at the time it was recorded. The fingerprint cannot be modified ex-post-facto (Immutability) and the proof will be stored permanently (Durability).

Kickstarter (Lighthouse)

Consistency + Atomicity + Integrity. If you sign one input and the output (Integrity) of a fundraiser transaction, others can contribute to the fundraiser but it cannot be spent (Atomicity) until the goal (output value) is funded (Consistency).

Payment Channels

Quorum of Control + Timelock + No Double Spend + Nonexpiration + Censorship Resistance + Authorization. A multisig 2-of-2 (Quorum) with a timelock (Timelock) used as the "settlement" transaction of a payment channel can be held (Nonexpiration) and spent at any time (Censorship Resistance) by either party (Authorization). The two parties can then create commitment transactions that double-spend (No Double-Spend) the settlement on a shorter timelock (Timelock).

Counterparty

Counterparty is a protocol layer built on top of bitcoin. The Counterparty protocol offers the ability to create and trade virtual assets and tokens. In addition, Counterparty offers a decentralized exchange for assets. Counterparty is also implementing smart contracts, based on the Ethereum Virtual Machine (EVM).

Counterparty embeds metadata in bitcoin transactions, using the OP_RETURN opcode or 1-of-N multisignature addresses that encode metadata in the place of public keys. Using these mechanisms, Counterparty implements a protocol layer encoded in bitcoin transactions. The additional protocol layer can be interpreted by applications that are Counterparty-aware, such as wallets and blockchain explorers, or any application built using the Counterparty libraries.

Counterparty can be used as a platform for other applications and services, in turn. For example, Tokenly is a platform built on top of Counterparty that allows content creators, artists, and companies to issue tokens that express digital ownership and can be used to rent, access, trade, or shop for content, products, and services. Other applications leveraging Counterparty include games (Spells of Genesis) and grid computing projects (Folding Coin).

More details about Counterparty can be found at https://counterparty.io. The open source project can be found at https://github.com/CounterpartyXCP.

Payment Channels and State Channels

Payment channels are a trustless mechanism for exchanging bitcoin transactions between two parties, outside of the bitcoin blockchain. These transactions, which would be valid if settled on the bitcoin blockchain, are held off-chain instead, acting as *promissory notes* for eventual batch settlement. Because the transactions are not settled, they can be exchanged without the usual settlement latency, allowing extremely high transaction throughput, low (submillisecond) latency, and fine (satoshi-level) granularity.

Actually, the term *channel* is a metaphor. State channels are virtual constructs represented by the exchange of state between two parties, outside of the blockchain. There are no "channels" per se and the underlying data transport mechanism is not the channel. We use the term channel to represent the relationship and shared state between two parties, outside of the blockchain.

To further explain this concept, think of a TCP stream. From the perspective of higher-level protocols it is a "socket" connecting two applications across the internet. But if you look at the network traffic, a TCP stream is just a virtual channel over IP packets. Each endpoint of the TCP stream sequences and assembles IP packets to create the illusion of a stream of bytes. Underneath, it’s all disconnected packets. Similarly, a payment channel is just a series of transactions. If properly sequenced and connected, they create redeemable obligations that you can trust even though you don’t trust the other side of the channel.

In this section we will look at various forms of payment channels. First, we will examine the mechanisms used to construct a one-way (unidirectional) payment channel for a metered micropayment service, such as streaming video. Then, we will expand on this mechanism and introduce bidirectional payment channels. Finally, we will look at how bidirectional channels can

be connected end-to-end to form multihop channels in a routed network, first proposed under the name *Lightning Network*.

Payment channels are part of the broader concept of a *state channel*, which represents an off-chain alteration of state, secured by eventual settlement in a blockchain. A payment channel is a state channel where the state being altered is the balance of a virtual currency.

State Channels—Basic Concepts and Terminology

A state channel is established between two parties, through a transaction that locks a shared state on the blockchain. This is called the *funding transaction* or *anchor transaction*. This single transaction must be transmitted to the network and mined to establish the channel. In the example of a payment channel, the locked state is the initial balance (in currency) of the channel.

The two parties then exchange signed transactions, called *commitment transactions*, that alter the initial state. These transactions are valid transactions in that they *could* be submitted for settlement by either party, but instead are held off-chain by each party pending the channel closure. State updates can be created as fast as each party can create, sign, and transmit a transaction to the other party. In practice this means that thousands of transactions per second can be exchanged.

When exchanging commitment transactions the two parties also invalidate the previous states, so that the most up-to-date commitment transaction is always the only one that can be redeemed. This prevents either party from cheating by unilaterally closing the channel with an expired prior state that is more favorable to them than the current state. We will examine the various mechanisms that can be used to invalidate prior state in the rest of this chapter.

Finally, the channel can be closed either cooperatively, by submitting a final *settlement transaction* to the blockchain, or unilaterally, by either party submitting the last commitment transaction to the blockchain. A unilateral close option is needed in case one of the parties unexpectedly disconnects. The settlement transaction represents the final state of the channel and is settled on the blockchain.

In the entire lifetime of the channel, only two transactions need to be submitted for mining on the blockchain: the funding and settlement transactions. In between these two states, the two parties can exchange any number of commitment transactions that are never seen by anyone else, nor submitted to the blockchain.

A payment channel between Bob and Alice, showing the funding, commitment, and settlement transactions illustrates a payment channel between Bob and Alice, showing the funding, commitment, and settlement transactions.

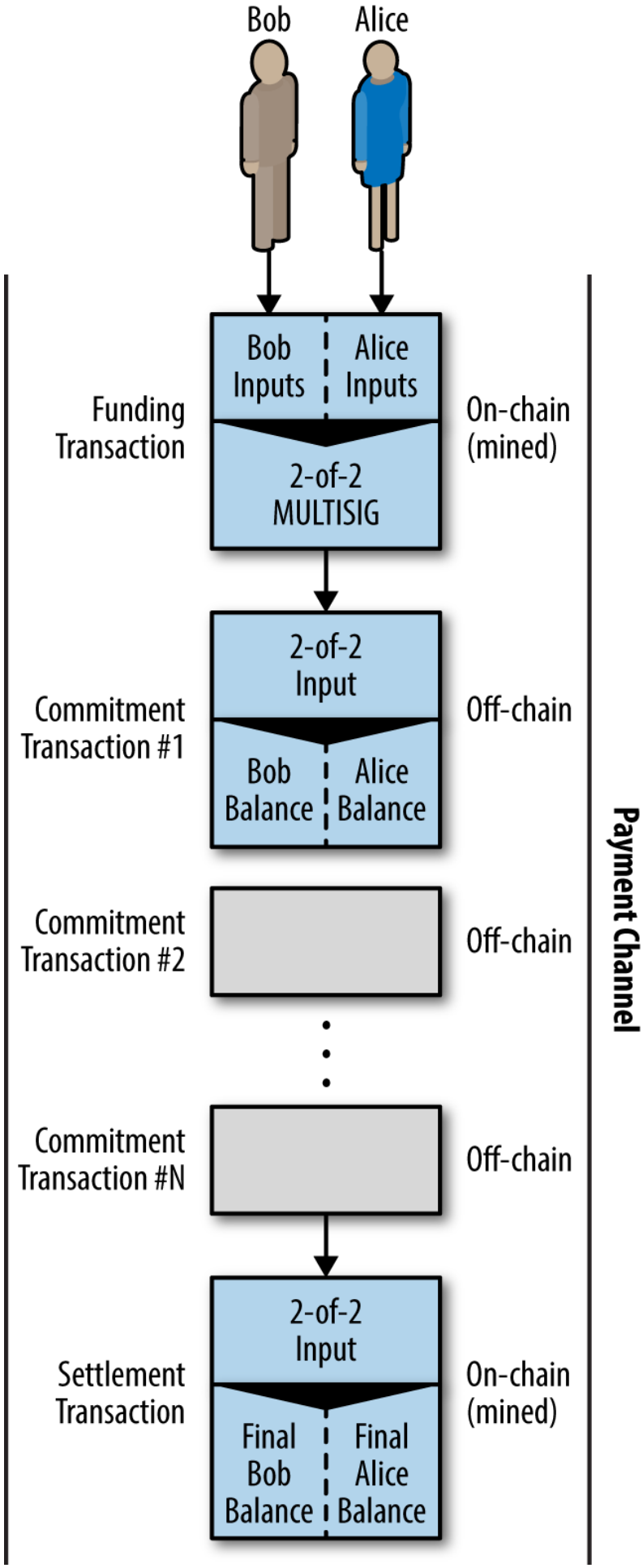

Figure 1. A payment channel between Bob and Alice, showing the funding, commitment, and settlement transactions

Simple Payment Channel Example

To explain state channels, we start with a very simple example. We demonstrate a one-way channel, meaning that value is flowing in one direction only. We will also start with the naive assumption that no one is trying to cheat, to keep things simple. Once we have the basic channel idea explained, we will then look at what it takes to make it trustless so that neither party *can* cheat, even if they are trying to.

For this example we will assume two participants: Emma and Fabian. Fabian offers a video streaming service that is billed by the second using a micropayment channel. Fabian charges 0.01 millibit (0.00001 BTC) per second of video, equivalent to 36 millibits (0.036 BTC) per hour of video. Emma is a user who purchases this streaming video service from Fabian. Emma purchases streaming video from Fabian with a payment channel, paying for each second of video shows Emma buying the video streaming service from Fabian using a payment channel.

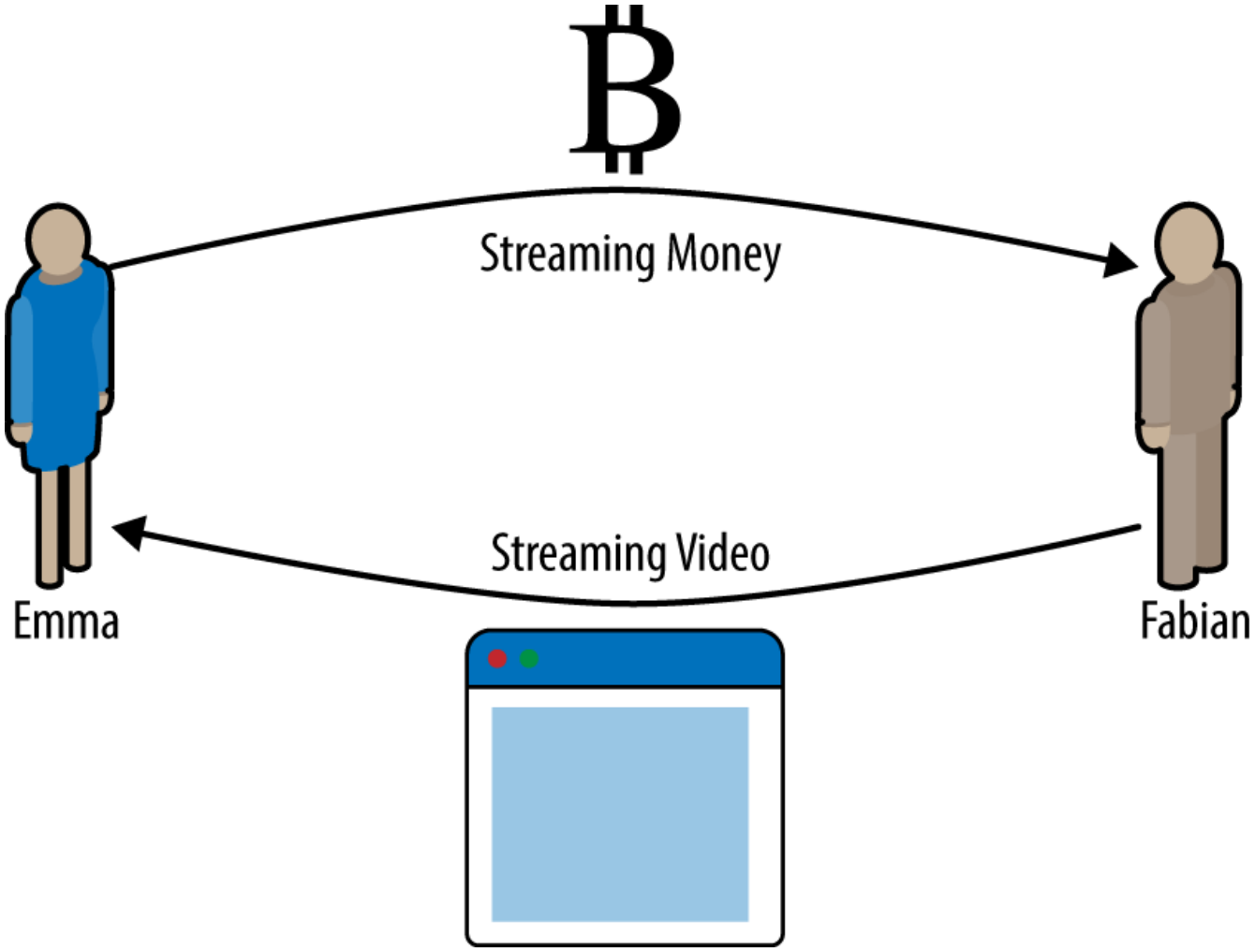

Figure 2. Emma purchases streaming video from Fabian with a payment channel, paying for each second of video

In this example, Fabian and Emma are using special software that handles both the payment channel and the video streaming. Emma is running the software in her browser, Fabian is running it on a server. The software includes basic bitcoin wallet functionality and can create and sign bitcoin transactions. Both the concept and the term "payment channel" are completely hidden from the users. What they see is video that is paid for by the second.

To set up the payment channel, Emma and Fabian establish a 2-of-2 multisignature address, with

each of them holding one of the keys. From Emma's perspective, the software in her browser presents a QR code with a P2SH address (starting with "3"), and asks her to submit a "deposit" for up to 1 hour of video. The address is then funded by Emma. Emma's transaction, paying to the multisignature address, is the funding or anchor transaction for the payment channel.

For this example, let's say that Emma funds the channel with 36 millibits (0.036 BTC). This will allow Emma to consume *up to* 1 hour of streaming video. The funding transaction in this case sets the maximum amount that can be transmitted in this channel, setting the *channel capacity*.

The funding transaction consumes one or more inputs from Emma's wallet, sourcing the funds. It creates one output with a value of 36 millibits paid to the multisignature 2-of-2 address controlled jointly between Emma and Fabian. It may have additional outputs for change back to Emma's wallet.

Once the funding transaction is confirmed, Emma can start streaming video. Emma's software creates and signs a commitment transaction that changes the channel balance to credit 0.01 millibit to Fabian's address and refund 35.99 millibits back to Emma. The transaction signed by Emma consumes the 36 millibits output created by the funding transaction and creates two outputs: one for her refund, the other for Fabian's payment. The transaction is only partially signed—it requires two signatures (2-of-2), but only has Emma's signature. When Fabian's server receives this transaction, it adds the second signature (for the 2-of-2 input) and returns it to Emma together with 1 second worth of video. Now both parties have a fully signed commitment transaction that either can redeem, representing the correct up-to-date balance of the channel. Neither party broadcasts this transaction to the network.

In the next round, Emma's software creates and signs another commitment transaction (commitment #2) that consumes the *same* 2-of-2 output from the funding transaction. The second commitment transaction allocates one output of 0.02 millibits to Fabian's address and one output of 35.98 millibits back to Emma's address. This new transaction is payment for two cumulative seconds of video. Fabian's software signs and returns the second commitment transaction, together with another second of video.

In this way, Emma's software continues to send commitment transactions to Fabian's server in exchange for streaming video. The balance of the channel gradually accumulates in favor of Fabian, as Emma consumes more seconds of video. Let's say Emma watches 600 seconds (10 minutes) of video, creating and signing 600 commitment transactions. The last commitment transaction (#600) will have two outputs, splitting the balance of the channel, 6 millibits to Fabian and 30 millibits to Emma.

Finally, Emma selects "Stop" to stop streaming video. Either Fabian or Emma can now transmit the final state transaction for settlement. This last transaction is the *settlement transaction* and pays Fabian for all the video Emma consumed, refunding the remainder of the funding transaction to Emma.

Emma's payment channel with Fabian, showing the commitment transactions that update the balance of the channel shows the channel between Emma and Fabian and the commitment transactions that update the balance of the channel.

In the end, only two transactions are recorded on the blockchain: the funding transaction that established the channel and a settlement transaction that allocated the final balance correctly

between the two participants.

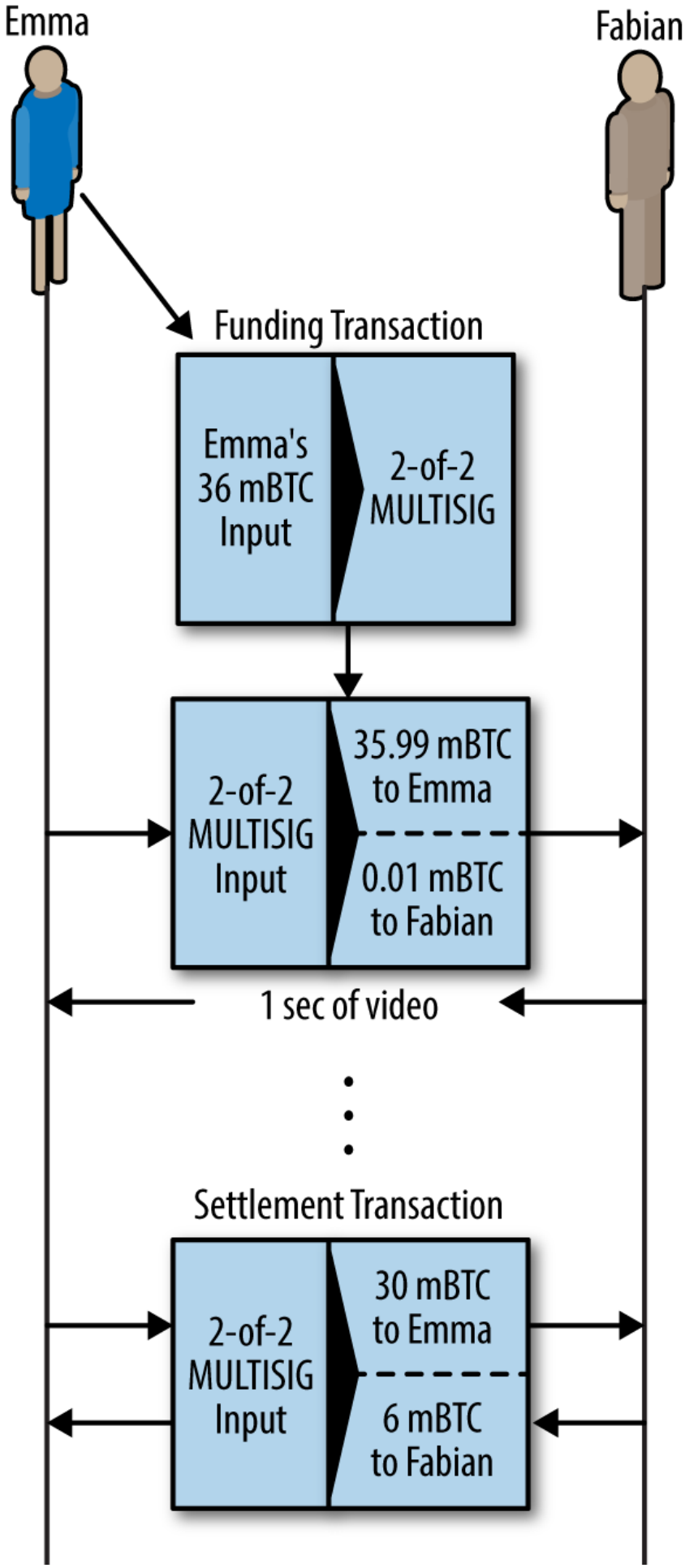

Figure 3. Emma's payment channel with Fabian, showing the commitment transactions that update the balance of the channel

Making Trustless Channels

The channel we just described works, but only if both parties cooperate, without any failures or attempts to cheat. Let's look at some of the scenarios that break this channel and see what is needed to fix those:

- Once the funding transaction happens, Emma needs Fabian's signature to get any money back. If Fabian disappears, Emma's funds are locked in a 2-of-2 and effectively lost. This channel, as constructed, leads to a loss of funds if one of the parties disconnects before there is at least one commitment transaction signed by both parties.
- While the channel is running, Emma can take any of the commitment transactions Fabian has countersigned and transmit one to the blockchain. Why pay for 600 seconds of video, if she can transmit commitment transaction #1 and only pay for 1 second of video? The channel fails because Emma can cheat by broadcasting a prior commitment that is in her favor.

Both of these problems can be solved with timelocks—let's look at how we could use transaction-level timelocks (nLocktime).

Emma cannot risk funding a 2-of-2 multisig unless she has a guaranteed refund. To solve this problem, Emma constructs the funding and refund transactions at the same time. She signs the funding transaction but doesn't transmit it to anyone. Emma transmits only the refund transaction to Fabian and obtains his signature.

The refund transaction acts as the first commitment transaction and its timelock establishes the upper bound for the channel's life. In this case, Emma could set the nLocktime to 30 days or 4320 blocks into the future. All subsequent commitment transactions must have a shorter timelock, so that they can be redeemed before the refund transaction.

Now that Emma has a fully signed refund transaction, she can confidently transmit the signed funding transaction knowing that she can eventually, after the timelock expires, redeem the refund transaction even if Fabian disappears.

Every commitment transaction the parties exchange during the life of the channel will be timelocked into the future. But the delay will be slightly shorter for each commitment so the most recent commitment can be redeemed before the prior commitment it invalidates. Because of the nLockTime, neither party can successfully propagate any of the commitment transactions until their timelock expires. If all goes well, they will cooperate and close the channel gracefully with a settlement transaction, making it unnecessary to transmit an intermediate commitment transaction. If not, the most recent commitment transaction can be propagated to settle the account and invalidate all prior commitment transactions.

For example, if commitment transaction #1 is timelocked to 4320 blocks in the future, then commitment transaction #2 is timelocked to 4319 blocks in the future. Commitment transaction #600 can be spent 600 blocks before commitment transaction #1 becomes valid.

Each commitment sets a shorter timelock, allowing it to be spent before the previous commitments become valid shows each commitment transaction setting a shorter timelock, allowing it to be spent before the previous commitments become valid.

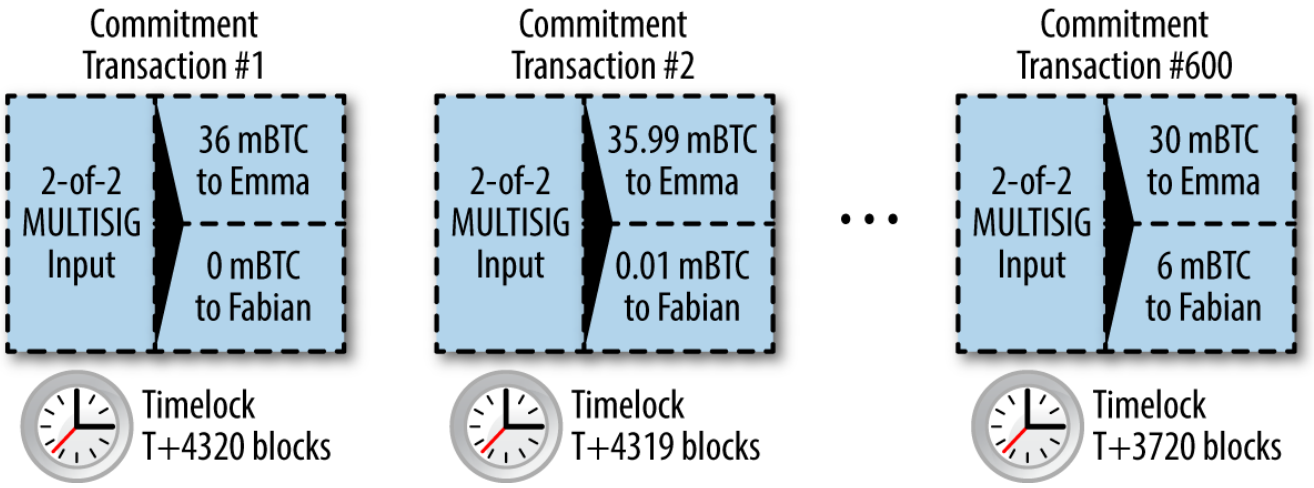

Figure 4. Each commitment sets a shorter timelock, allowing it to be spent before the previous commitments become valid

Each subsequent commitment transaction must have a shorter timelock so that it may be broadcast before its predecessors and before the refund transaction. The ability to broadcast a commitment earlier ensures it will be able to spend the funding output and preclude any other commitment transaction from being redeemed by spending the output. The guarantees offered by the bitcoin blockchain, preventing double-spends and enforcing timelocks, effectively allow each commitment transaction to invalidate its predecessors.

State channels use timelocks to enforce smart contracts across a time dimension. In this example we saw how the time dimension guarantees that the most recent commitment transaction becomes valid before any earlier commitments. Thus, the most recent commitment transaction can be transmitted, spending the inputs and invalidating prior commitment transactions. The enforcement of smart contracts with absolute timelocks protects against cheating by one of the parties. This implementation needs nothing more than absolute transaction-level timelocks (nLocktime). Next, we will see how script-level timelocks, CHECKLOCKTIMEVERIFY and CHECKSEQUENCEVERIFY, can be used to construct more flexible, useful, and sophisticated state channels.

The first form of unidirectional payment channel was demonstrated as a prototype video streaming application in 2015 by an Argentinian team of developers.

Timelocks are not the only way to invalidate prior commitment transactions. In the next sections we will see how a revocation key can be used to achieve the same result. Timelocks are effective but they have two distinct disadvantages. By establishing a maximum timelock when the channel is first opened, they limit the lifetime of the channel. Worse, they force channel implementations to strike a balance between allowing long-lived channels and forcing one of the participants to wait a very long time for a refund in case of premature closure. For example, if you allow the channel to remain open for 30 days, by setting the refund timelock to 30 days, if one of the parties disappears immediately the other party must wait 30 days for a refund. The more distant the endpoint, the more distant the refund.

The second problem is that since each subsequent commitment transaction must decrement the timelock, there is an explicit limit on the number of commitment transactions that can be exchanged between the parties. For example, a 30-day channel, setting a timelock of 4320 blocks into the future, can only accommodate 4320 intermediate commitment transactions before it must be closed. There is a danger in setting the timelock commitment transaction interval at 1 block. By

setting the timelock interval between commitment transactions to 1 block, a developer is creating a very high burden for the channel participants who have to be vigilant, remain online and watching, and be ready to transmit the right commitment transaction at any time.

Now that we understand how timelocks can be used to invalidate prior commitments, we can see the difference between closing the channel cooperatively and closing it unilaterally by broadcasting a commitment transaction. All commitment transactions are timelocked, therefore broadcasting a commitment transaction will always involve waiting until the timelock has expired. But if the two parties agree on what the final balance is and know they both hold commitment transactions that will eventually make that balance a reality, they can construct a settlement transaction without a timelock representing that same balance. In a cooperative close, either party takes the most recent commitment transaction and builds a settlement transaction that is identical in every way except that it omits the timelock. Both parties can sign this settlement transaction knowing there is no way to cheat and get a more favorable balance. By cooperatively signing and transmitting the settlement transaction they can close the channel and redeem their balance immediately. Worst case, one of the parties can be petty, refuse to cooperate, and force the other party to do a unilateral close with the most recent commitment transaction. But if they do that, they have to wait for their funds too.

Asymmetric Revocable Commitments

A better way to handle the prior commitment states is to explicitly revoke them. However, this is not easy to achieve. A key characteristic of bitcoin is that once a transaction is valid, it remains valid and does not expire. The only way to cancel a transaction is by double-spending its inputs with another transaction before it is mined. That's why we used timelocks in the simple payment channel example above to ensure that more recent commitments could be spent before older commitments were valid. However, sequencing commitments in time creates a number of constraints that make payment channels difficult to use.

Even though a transaction cannot be canceled, it can be constructed in such a way as to make it undesirable to use. The way we do that is by giving each party a *revocation key* that can be used to punish the other party if they try to cheat. This mechanism for revoking prior commitment transactions was first proposed as part of the Lightning Network.

To explain revocation keys, we will construct a more complex payment channel between two exchanges run by Hitesh and Irene. Hitesh and Irene run bitcoin exchanges in India and the USA, respectively. Customers of Hitesh's Indian exchange often send payments to customers of Irene's USA exchange and vice versa. Currently, these transactions occur on the bitcoin blockchain, but this means paying fees and waiting several blocks for confirmations. Setting up a payment channel between the exchanges will significantly reduce the cost and accelerate the transaction flow.

Hitesh and Irene start the channel by collaboratively constructing a funding transaction, each funding the channel with 5 bitcoin. The initial balance is 5 bitcoin for Hitesh and 5 bitcoin for Irene. The funding transaction locks the channel state in a 2-of-2 multisig, just like in the example of a simple channel.

The funding transaction may have one or more inputs from Hitesh (adding up to 5 bitcoin or more), and one or more inputs from Irene (adding up to 5 bitcoin or more). The inputs have to slightly exceed the channel capacity in order to cover the transaction fees. The transaction has one output that locks the 10 total bitcoin to a 2-of-2 multisig address controlled by both Hitesh and Irene. The

funding transaction may also have one or more outputs returning change to Hitesh and Irene if their inputs exceeded their intended channel contribution. This is a single transaction with inputs offered and signed by two parties. It has to be constructed in collaboration and signed by each party before it is transmitted.

Now, instead of creating a single commitment transaction that both parties sign, Hitesh and Irene create two different commitment transactions that are *asymmetric*.

Hitesh has a commitment transaction with two outputs. The first output pays Irene the 5 bitcoin she is owed *immediately*. The second output pays Hitesh the 5 bitcoin he is owed, but only after a timelock of 1000 blocks. The transaction outputs look like this:

```
Input: 2-of-2 funding output, signed by Irene

Output 0 <5 bitcoin>:
    <Irene's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Hitesh's Public Key> CHECKSIG
```

Irene has a different commitment transaction with two outputs. The first output pays Hitesh the 5 bitcoin he is owed immediately. The second output pays Irene the 5 bitcoin she is owed but only after a timelock of 1000 blocks. The commitment transaction Irene holds (signed by Hitesh) looks like this:

```
Input: 2-of-2 funding output, signed by Hitesh

Output 0 <5 bitcoin>:
    <Hitesh's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Irene's Public Key> CHECKSIG
```

This way, each party has a commitment transaction, spending the 2-of-2 funding output. This input is signed by the *other* party. At any time the party holding the transaction can also sign (completing the 2-of-2) and broadcast. However, if they broadcast the commitment transaction, it pays the other party immediately whereas they have to wait for a timelock to expire. By imposing a delay on the redemption of one of the outputs, we put each party at a slight disadvantage when they choose to unilaterally broadcast a commitment transaction. But a time delay alone isn't enough to encourage fair conduct.

Two asymmetric commitment transactions with delayed payment for the party holding the

transaction shows two asymmetric commitment transactions, where the output paying the holder of the commitment is delayed.

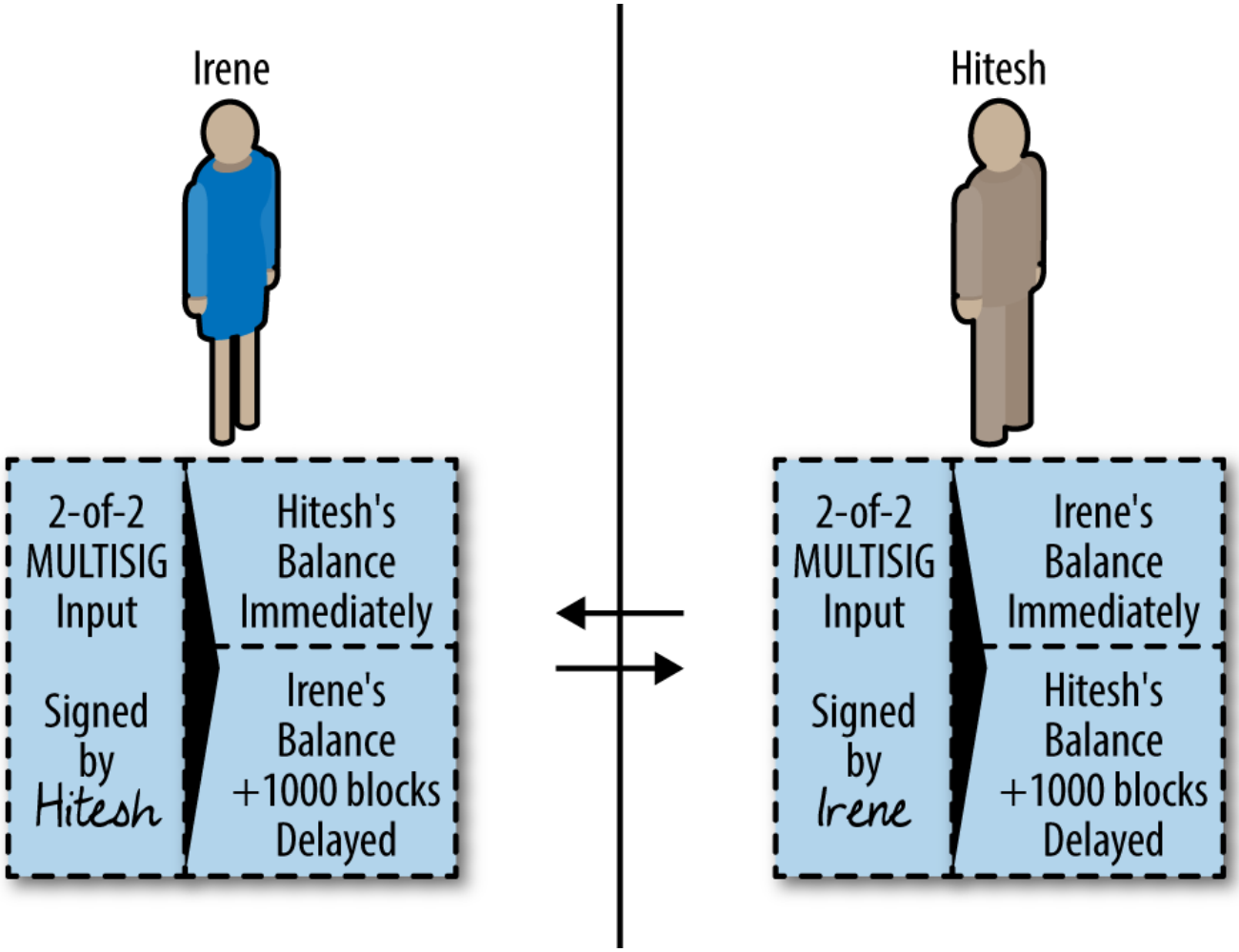

Figure 5. Two asymmetric commitment transactions with delayed payment for the party holding the transaction

Now we introduce the final element of this scheme: a revocation key that prevents a cheater from broadcasting an expired commitment. The revocation key allows the wronged party to punish the cheater by taking the entire balance of the channel.

The revocation key is composed of two secrets, each half generated independently by each channel participant. It is similar to a 2-of-2 multisig, but constructed using elliptic curve arithmetic, so that both parties know the revocation public key but each party knows only half the revocation secret key.

In each round, both parties reveal their half of the revocation secret to the other party, thereby giving the other party (who now has both halves) the means to claim the penalty output if this revoked transaction is ever broadcast.

Each of the commitment transactions has a "delayed" output. The redemption script for that output allows one party to redeem it after 1000 blocks, *or* the other party to redeem it if they have a revocation key, penalizing transmission of a revoked commitment.

So when Hitesh creates a commitment transaction for Irene to sign, he makes the second output payable to himself after 1000 blocks, or to the revocation public key (of which he only knows half

the secret). Hitesh constructs this transaction. He will only reveal his half of the revocation secret to Irene when he is ready to move to a new channel state and wants to revoke this commitment.

The second output's script looks like this:

```
Output 0 <5 bitcoin>:
    <Irene's Public Key> CHECKSIG

Output 1 <5 bitcoin>:
IF
    # Revocation penalty output
    <Revocation Public Key>
ELSE
    <1000 blocks>
    CHECKSEQUENCEVERIFY
    DROP
    <Hitesh's Public Key>
ENDIF
CHECKSIG
```

Irene can confidently sign this transaction, since if transmitted it will immediately pay her what she is owed. Hitesh holds the transaction, but knows that if he transmits it in a unilateral channel closing, he will have to wait 1000 blocks to get paid.

When the channel is advanced to the next state, Hitesh has to *revoke* this commitment transaction before Irene agrees to sign the next commitment transaction. To do that, all he has to do is send his half of the *revocation key* to Irene. Once Irene has both halves of the revocation secret key for this commitment, she can sign the next commitment with confidence. She knows that if Hitesh tries to cheat by publishing the prior commitment, she can use the revocation key to redeem Hitesh's delayed output. *If Hitesh cheats, Irene gets BOTH outputs*. Meanwhile, Hitesh only has half the revocation secret for that revocation public key and can't redeem the output until 1000 blocks. Irene will be able to redeem the output and punish Hitesh before the 1000 blocks have elapsed.

The revocation protocol is bilateral, meaning that in each round, as the channel state is advanced, the two parties exchange new commitments, exchange revocation secrets for the previous commitments, and sign each other's new commitment transactions. As they accept a new state, they make the prior state impossible to use, by giving each other the necessary revocation secrets to punish any cheating.

Let's look at an example of how it works. One of Irene's customers wants to send 2 bitcoin to one of Hitesh's customers. To transmit 2 bitcoin across the channel, Hitesh and Irene must advance the channel state to reflect the new balance. They will commit to a new state (state number 2) where the channel's 10 bitcoin are split, 7 bitcoin to Hitesh and 3 bitcoin to Irene. To advance the state of the channel, they will each create new commitment transactions reflecting the new channel balance.

As before, these commitment transactions are asymmetric so that the commitment transaction each party holds forces them to wait if they redeem it. Crucially, before signing new commitment transactions, they must first exchange revocation keys to invalidate the prior commitment. In this

particular case, Hitesh's interests are aligned with the real state of the channel and therefore he has no reason to broadcast a prior state. However, for Irene, state number 1 leaves her with a higher balance than state 2. When Irene gives Hitesh the revocation key for her prior commitment transaction (state number 1) she is effectively revoking her ability to profit from regressing the channel to a prior state because with the revocation key, Hitesh can redeem both outputs of the prior commitment transaction without delay. Meaning if Irene broadcasts the prior state, Hitesh can exercise his right to take all of the outputs.

Importantly, the revocation doesn't happen automatically. While Hitesh has the ability to punish Irene for cheating, he has to watch the blockchain diligently for signs of cheating. If he sees a prior commitment transaction broadcast, he has 1000 blocks to take action and use the revocation key to thwart Irene's cheating and punish her by taking the entire balance, all 10 bitcoin.

Asymmetric revocable commitments with relative time locks (CSV) are a much better way to implement payment channels and a very significant innovation in this technology. With this construct, the channel can remain open indefinitely and can have billions of intermediate commitment transactions. In prototype implementations of Lightning Network, the commitment state is identified by a 48-bit index, allowing more than 281 trillion (2.8×10^{14}) state transitions in any single channel!

Hash Time Lock Contracts (HTLC)

Payment channels can be further extended with a special type of smart contract that allows the participants to commit funds to a redeemable secret, with an expiration time. This feature is called a *Hash Time Lock Contract*, or *HTLC*, and is used in both bidirectional and routed payment channels.

Let's first explain the "hash" part of the HTLC. To create an HTLC, the intended recipient of the payment will first create a secret R. They then calculate the hash of this secret H:

```
H = Hash(R)
```

This produces a hash H that can be included in an output's locking script. Whoever knows the secret can use it to redeem the output. The secret R is also referred to as a *preimage* to the hash function. The preimage is just the data that is used as input to a hash function.

The second part of an HTLC is the "time lock" component. If the secret is not revealed, the payer of the HTLC can get a "refund" after some time. This is achieved with an absolute time lock using CHECKLOCKTIMEVERIFY.

The script implementing an HTLC might look like this:

```
IF
    # Payment if you have the secret R
    HASH160 <H> EQUALVERIFY
ELSE
    # Refund after timeout.
    <locktime> CHECKLOCKTIMEVERIFY DROP
    <Payer Public Key> CHECKSIG
ENDIF
```

Anyone who knows the secret R, which when hashed equals to H, can redeem this output by exercising the first clause of the IF flow.

If the secret is not revealed and the HTLC claimed, after a certain number of blocks the payer can claim a refund using the second clause in the IF flow.

This is a basic implementation of an HTLC. This type of HTLC can be redeemed by *anyone* who has the secret R. An HTLC can take many different forms with slight variations to the script. For example, adding a CHECKSIG operator and a public key in the first clause restricts redemption of the hash to a named recipient, who must also know the secret R.

Routed Payment Channels (Lightning Network)

The Lightning Network is a proposed routed network of bidirectional payment channels connected end-to-end. A network like this can allow any participant to route a payment from channel to channel without trusting any of the intermediaries. The Lightning Network was first described by Joseph Poon and Thadeus Dryja in February 2015, building on the concept of payment channels as proposed and elaborated upon by many others.

"Lightning Network" refers to a specific design for a routed payment channel network, which has now been implemented by at least five different open source teams. The independent implementations are coordinated by a set of interoperability standards described in the *Basics of Lightning Technology (BOLT)* paper.

Prototype implementations of the Lightning Network have been released by several teams.

The Lightning Network is one possible way of implementing routed payment channels. There are several other designs that aim to achieve similar goals, such as Teechan and Tumblebit.

Basic Lightning Network Example

Let's see how this works.

In this example, we have five participants: Alice, Bob, Carol, Diana, and Eric. These five participants have opened payment channels with each other, in pairs. Alice has a payment channel with Bob. Bob is connected to Carol, Carol to Diana, and Diana to Eric. For simplicity let's assume each channel is funded with 2 bitcoin by each participant, for a total capacity of 4 bitcoin in each channel.

A series of bidirectional payment channels linked to form a Lightning Network that can route a payment from Alice to Eric shows five participants in a Lightning Network, connected by bidirectional payment channels that can be linked to make a payment from Alice to Eric (Routed Payment Channels (Lightning Network)).

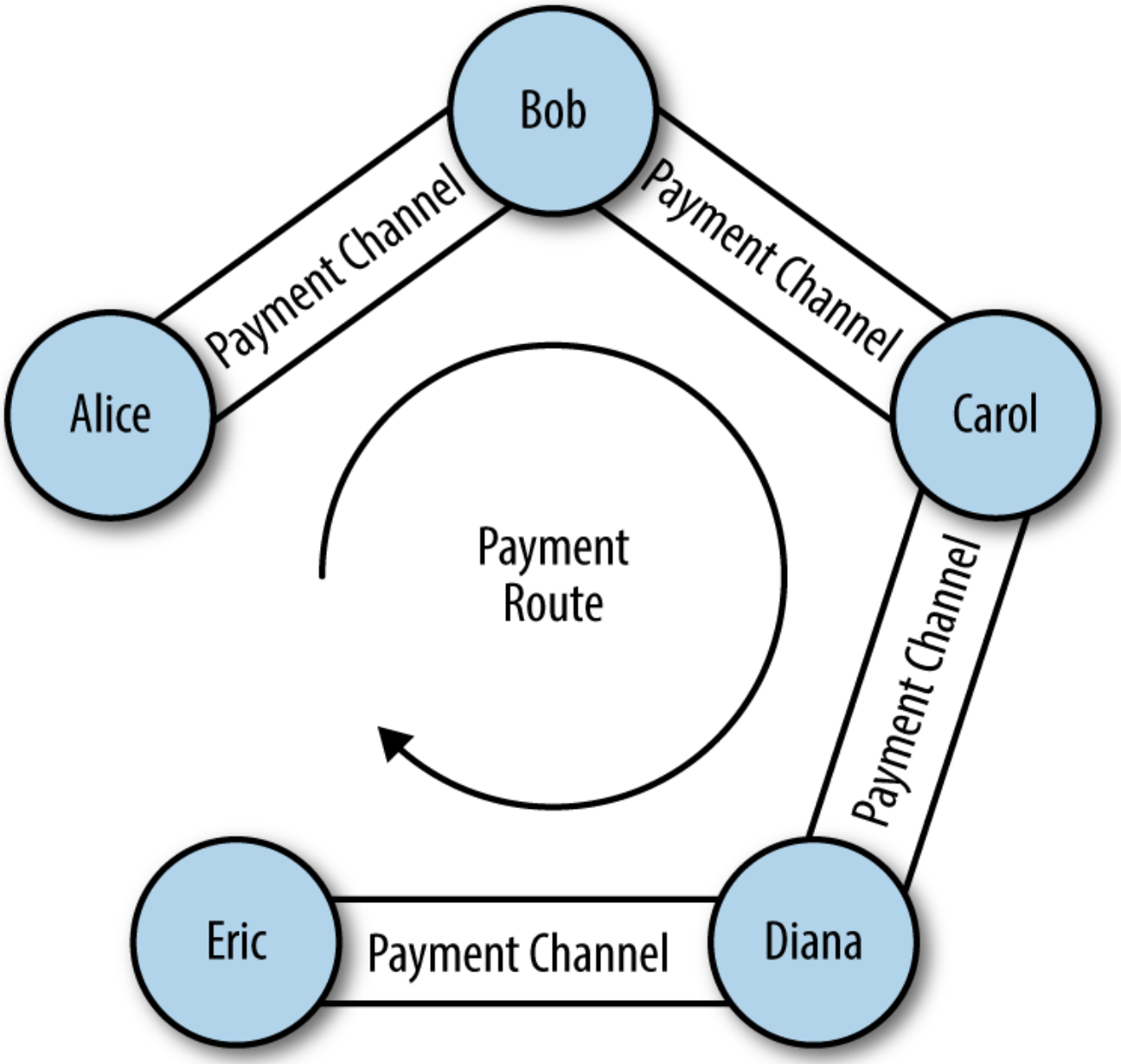

Figure 6. A series of bidirectional payment channels linked to form a Lightning Network that can route a payment from Alice to Eric

Alice wants to pay Eric 1 bitcoin. However, Alice is not connected to Eric by a payment channel. Creating a payment channel requires a funding transaction, which must be committed to the bitcoin blockchain. Alice does not want to open a new payment channel and commit more of her funds. Is there a way to pay Eric, indirectly?

Step-by-step payment routing through a Lightning Network shows the step-by-step process of routing a payment from Alice to Eric, through a series of HTLC commitments on the payment channels connecting the participants.

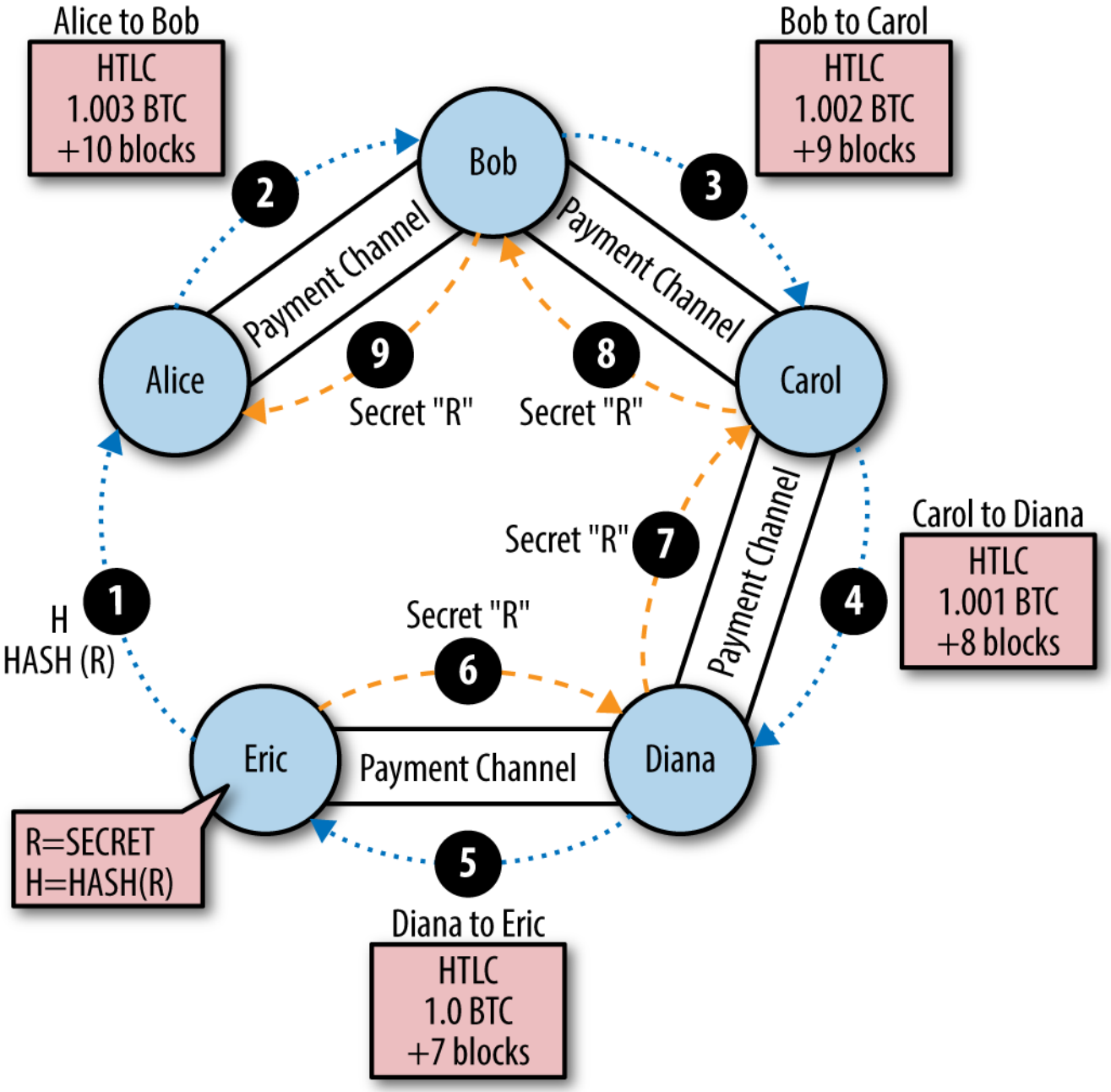

Figure 7. Step-by-step payment routing through a Lightning Network

Alice is running a Lightning Network (LN) node that is keeping track of her payment channel to Bob and has the ability to discover routes between payment channels. Alice's LN node also has the ability to connect over the internet to Eric's LN node. Eric's LN node creates a secret R using a random number generator. Eric's node does not reveal this secret to anyone. Instead, Eric's node calculates a hash H of the secret R and transmits this hash to Alice's node (see Step-by-step payment routing through a Lightning Network step 1).

Now Alice's LN node constructs a route between Alice's LN node and Eric's LN node. The routing algorithm used will be examined in more detail later, but for now let's assume that Alice's node can find an efficient route.

Alice's node then constructs an HTLC, payable to the hash H, with a 10-block refund timeout (current block + 10), for an amount of 1.003 bitcoin (see Step-by-step payment routing through a Lightning Network step 2). The extra 0.003 will be used to compensate the intermediate nodes for their participation in this payment route. Alice offers this HTLC to Bob, deducting 1.003 bitcoin

from her channel balance with Bob and committing it to the HTLC. The HTLC has the following meaning: *"Alice is committing 1.003 of her channel balance to be paid to Bob if Bob knows the secret, or refunded back to Alice's balance if 10 blocks elapse."* The channel balance between Alice and Bob is now expressed by commitment transactions with three outputs: 2 bitcoin balance to Bob, 0.997 bitcoin balance to Alice, 1.003 bitcoin committed in Alice's HTLC. Alice's balance is reduced by the amount committed to the HTLC.

Bob now has a commitment that if he is able to get the secret R within the next 10 blocks, he can claim the 1.003 locked by Alice. With this commitment in hand, Bob's node constructs an HTLC on his payment channel with Carol. Bob's HTLC commits 1.002 bitcoin to hash H for 9 blocks, which Carol can redeem if she has secret R (see Step-by-step payment routing through a Lightning Network step 3). Bob knows that if Carol can claim his HTLC, she has to produce R. If Bob has R in nine blocks, he can use it to claim Alice's HTLC to him. He also makes 0.001 bitcoin for committing his channel balance for nine blocks. If Carol is unable to claim his HTLC and he is unable to claim Alice's HTLC, everything reverts back to the prior channel balances and no one is at a loss. The channel balance between Bob and Carol is now: 2 to Carol, 0.998 to Bob, 1.002 committed by Bob to the HTLC.

Carol now has a commitment that if she gets R within the next nine blocks, she can claim 1.002 bitcoin locked by Bob. Now she can make an HTLC commitment on her channel with Diana. She commits an HTLC of 1.001 bitcoin to hash H, for eight blocks, which Diana can redeem if she has secret R (see Step-by-step payment routing through a Lightning Network step 4). From Carol's perspective, if this works she is 0.001 bitcoin better off and if it doesn't she loses nothing. Her HTLC to Diana is only viable if R is revealed, at which point she can claim the HTLC from Bob. The channel balance between Carol and Diana is now: 2 to Diana, 0.999 to Carol, 1.001 committed by Carol to the HTLC.

Finally, Diana can offer an HTLC to Eric, committing 1 bitcoin for seven blocks to hash H (see Step-by-step payment routing through a Lightning Network step 5). The channel balance between Diana and Eric is now: 2 to Eric, 1 to Diana, 1 committed by Diana to the HTLC.

However, at this hop in the route, Eric *has* secret R. He can therefore claim the HTLC offered by Diana. He sends R to Diana and claims the 1 bitcoin, adding it to his channel balance (see Step-by-step payment routing through a Lightning Network step 6). The channel balance is now: 1 to Diana, 3 to Eric.

Now, Diana has secret R. Therefore, she can now claim the HTLC from Carol. Diana transmits R to Carol and adds the 1.001 bitcoin to her channel balance (see Step-by-step payment routing through a Lightning Network step 7). Now the channel balance between Carol and Diana is: 0.999 to Carol, 3.001 to Diana. Diana has "earned" 0.001 for participating in this payment route.

Flowing back through the route, the secret R allows each participant to claim the outstanding HTLCs. Carol claims 1.002 from Bob, setting the balance on their channel to: 0.998 to Bob, 3.002 to Carol (see Step-by-step payment routing through a Lightning Network step 8). Finally, Bob claims the HTLC from Alice (see Step-by-step payment routing through a Lightning Network step 9). Their channel balance is updated as: 0.997 to Alice, 3.003 to Bob.

Alice has paid Eric 1 bitcoin without opening a channel to Eric. None of the intermediate parties in the payment route had to trust each other. For the short-term commitment of their funds in the

channel they are able to earn a small fee, with the only risk being a small delay in refund if the channel was closed or the routed payment failed.

Lightning Network Transport and Routing

All communications between LN nodes are encrypted point-to-point. In addition, nodes have a long-term public key that they use as an identifier and to authenticate each other.

Whenever a node wishes to send a payment to another node, it must first construct a *path* through the network by connecting payment channels with sufficient capacity. Nodes advertise routing information, including what channels they have open, how much capacity each channel has, and what fees they charge to route payments. The routing information can be shared in a variety of ways and different routing protocols are likely to emerge as Lightning Network technology advances. Some Lightning Network implementations use the IRC protocol as a convenient mechanism for nodes to announce routing information. Another implementation of route discovery uses a P2P model where nodes propagate channel announcements to their peers, in a "flooding" model, similar to how bitcoin propagates transactions. Future plans include a proposal called Flare, which is a hybrid routing model with local node "neighborhoods" and longer-range beacon nodes.

In our previous example, Alice's node uses one of these route discovery mechanisms to find one or more paths connecting her node to Eric's node. Once Alice's node has constructed a path, she will initialize that path through the network, by propagating a series of encrypted and nested instructions to connect each of the adjacent payment channels.

Importantly, this path is only known to Alice's node. All other participants in the payment route see only the adjacent nodes. From Carol's perspective, this looks like a payment from Bob to Diana. Carol does not know that Bob is actually relaying a payment from Alice. She also doesn't know that Diana will be relaying a payment to Eric.

This is a critical feature of the Lightning Network, because it ensures privacy of payments and makes it very difficult to apply surveillance, censorship, or blacklists. But how does Alice establish this payment path, without revealing anything to the intermediary nodes?

The Lightning Network implements an onion-routed protocol based on a scheme called Sphinx. This routing protocol ensures that a payment sender can construct and communicate a path through the Lightning Network such that:

- Intermediate nodes can verify and decrypt their portion of route information and find the next hop.
- Other than the previous and next hops, they cannot learn about any other nodes that are part of the path.
- They cannot identify the length of the payment path, or their own position in that path.
- Each part of the path is encrypted in such a way that a network-level attacker cannot associate the packets from different parts of the path to each other.
- Unlike Tor (an onion-routed anonymization protocol on the internet), there are no "exit nodes" that can be placed under surveillance. The payments do not need to be transmitted to the bitcoin blockchain; the nodes just update channel balances.

Using this onion-routed protocol, Alice wraps each element of the path in a layer of encryption, starting with the end and working backward. She encrypts a message to Eric with Eric's public key. This message is wrapped in a message encrypted to Diana, identifying Eric as the next recipient. The message to Diana is wrapped in a message encrypted to Carol's public key and identifying Diana as the next recipient. The message to Carol is encrypted to Bob's key. Thus, Alice has constructed this encrypted multilayer "onion" of messages. She sends this to Bob, who can only decrypt and unwrap the outer layer. Inside, Bob finds a message addressed to Carol that he can forward to Carol but cannot decipher himself. Following the path, the messages get forwarded, decrypted, forwarded, etc., all the way to Eric. Each participant knows only the previous and next node in each hop.

Each element of the path contains information on the HTLC that must be extended to the next hop, the amount that is being sent, the fee to include, and the CLTV locktime (in blocks) expiration of the HTLC. As the route information propagates, the nodes make HTLC commitments forward to the next hop.

At this point, you might be wondering how it is possible that the nodes do not know the length of the path and their position in that path. After all, they receive a message and forward it to the next hop. Doesn't it get shorter, allowing them to deduce the path size and their position? To prevent this, the path is always fixed at 20 hops and padded with random data. Each node sees the next hop and a fixed-length encrypted message to forward. Only the final recipient sees that there is no next hop. To everyone else it seems as if there are always 20 more hops to go.

Lightning Network Benefits

A Lightning Network is a second-layer routing technology. It can be applied to any blockchain that supports some basic capabilities, such as multisignature transactions, timelocks, and basic smart contracts.

If a Lightning Network is layered on top of the bitcoin network, the bitcoin network can gain a significant increase in capacity, privacy, granularity, and speed, without sacrificing the principles of trustless operation without intermediaries:

Privacy

Lightning Network payments are much more private than payments on the bitcoin blockchain, as they are not public. While participants in a route can see payments propagated across their channels, they do not know the sender or recipient.

Fungibility

A Lightning Network makes it much more difficult to apply surveillance and blacklists on bitcoin, increasing the fungibility of the currency.

Speed

Bitcoin transactions using Lightning Network are settled in milliseconds, rather than minutes, as HTLCs are cleared without committing transactions to a block.

Granularity

A Lightning Network can enable payments at least as small as the bitcoin "dust" limit, perhaps even smaller. Some proposals allow for subsatoshi increments.

Capacity

A Lightning Network increases the capacity of the bitcoin system by several orders of magnitude. There is no practical upper bound to the number of payments per second that can be routed over a Lightning Network, as it depends only on the capacity and speed of each node.

Trustless Operation

A Lightning Network uses bitcoin transactions between nodes that operate as peers without trusting each other. Thus, a Lightning Network preserves the principles of the bitcoin system, while expanding its operating parameters significantly.

Of course, as mentioned previously, the Lightning Network protocol is not the only way to implement routed payment channels. Other proposed systems include Tumblebit and Teechan. At this time, however, the Lightning Network has already been deployed on testnet. Several different teams have developed competing implementations of LN and are working toward a common interoperability standard (called BOLT). It is likely that Lightning Network will be the first routed payment channel network to be deployed in production.

Conclusion

We have examined just a few of the emerging applications that can be built using the bitcoin blockchain as a trust platform. These applications expand the scope of bitcoin beyond payments and beyond financial instruments, to encompass many other applications where trust is critical. By decentralizing the basis of trust, the bitcoin blockchain is a platform that will spawn many revolutionary applications in a wide variety of industries.