Deeper Network: Tech Paper

Proposed structure

Introduction

Goals of the system

Before coming up with a solution, we first need to define the problems we are trying to resolve. e^2 chain is the infrastructure of Deeper Network for Cyber Security and Information Privacy, which is empowered by sharing and collaborations. All participants in Deeper Network should be able to get involved in the sharing economy, blockchain activity (provided they meet the minimum credit score requirement), and Always On Information Privacy service. DApp developers can also create distributed applications such as digital currency trading, social network service (SNS), and e-commerce platforms on the e^2 chain.

Given these requirements, the e^2 chain need to be low energy consumption. Unlike the standard Nakamoto consensus protocols, our DPoC does not rely on solving a significant computation puzzle to vote for consensus, instead your credit represents your vote. The computation cost of our consensus is ignorable compared to any other functionalities provided and applications that will be supported on the blockchain.

e^2 chain is providing Information Privacy protection for all participants, by connecting everyone together. Similar to Tor Project, all participants set up peer to peer encrypted links to protect privacy for each other. These valuable collaborations are empowered and motivated by sharing economy. The economy mode intrinsically will generate a good amount of transactions.

Solution structure

- Layers from the WP (decomposing the structure of the paper: hardware, OS, networking, consensus, tokenomics, application layer)

Deeper Connect is an all-in-one solution combining software and hardware. The software consists of a data plane, a management plane and a control plane. The data plane, implemented with Deeper's independently developed AtomOS, is responsible for handling user data packet transmission, reception and deep inspection. The function of the management plane is to provide a user-friendly interface for monitoring system operations or changing system configurations. The control plane handles communication between device and blockchain, communication between devices, and supports the blockchain consensus mechanism. Our hardware is the Intel Atom embedded processor. The principles of the software framework are easily grasped in a layered architecture view, as shown in Figure 10.

Hardware

Key points:

- Deeper builds its own hardware solution, a router device that gets inserted between the WAN cable and the user's PC (confirm / deny)
- It's built on (zzz), standard hardware components (if it does)
- There is a VN implementation

Assets:

- Specs from the WP
- Notes about encrypted drives

The Deeper project's goal is to provide plug-and-play hardware to solve difficult tech- nical issues in the area of security, sharing and blockchain – an all-in-one solution. The unique features of Deeper Connect's hardware design will be shown as follows.

Low Energy Consumption

According to digiconomist's assessment [4], the accumulated total energy consumption of bitcoin mining worldwide reached 188 million kWh as of May 25, 2018, meaning annual energy consumption is 68.81 billion kWh; six times of the energy consumption for May 2017 (11.57 billion kWh).

The energy consumption of all bitcoin miners around the world is equivalent to the energy consumption of the Czech Republic, which is 0.31% of global energy consump- tion. The average energy consumption for each bitcoin transaction is 968 kWh, the same as the energy consumption of 32 U.S. families in one day. Currently, Bitcoin's annual carbon emissions amount to 33.85 million tons, or 1,300 kilograms of carbon per bitcoin [45].

The unique design of Deeper's PoC consensus algorithm solves this issue by en- abling mining rigs to participate in consensus networks with extremely low computing resources. Deeper Connect utilizes low consumption embedded processors to build a consensus network and network sharing. The maximum energy consumption of a Deeper Connect is 15 watts. In the future, a Deeper Connect Mini with even lower energy consumption (maximum 5W) will be introduced.

Deeper hardware running AtomOS is the most energy efficient security product on the market. Deeper Connect has the potential to become the most profitable blockchain mining rig.

Hardware Type	**Energy Consumption**

Deeper Connect	5-15W
ASIC mining rig	2,000-3,000W
GPU mining rig	1,000-2,000W

Table: Mining Rig Energy Consumption Comparison

Hardware Wallet for Cryptocurrency

Deeper's security hardware also integrates a cryptocurrency wallet feature to pro- vide users with the highest level of cryptocurrency security without any knowledge of blockchain or network security.

Deeper Connect provides multiple security guarantees with AtomOS, which makes it impossible for crackers and malicious organizations to remotely obtain control of the hardware. As a result, the key information stored in the device is inaccessible to crackers. Additionally, malicious attacks will be identified and recorded to help catch crackers.

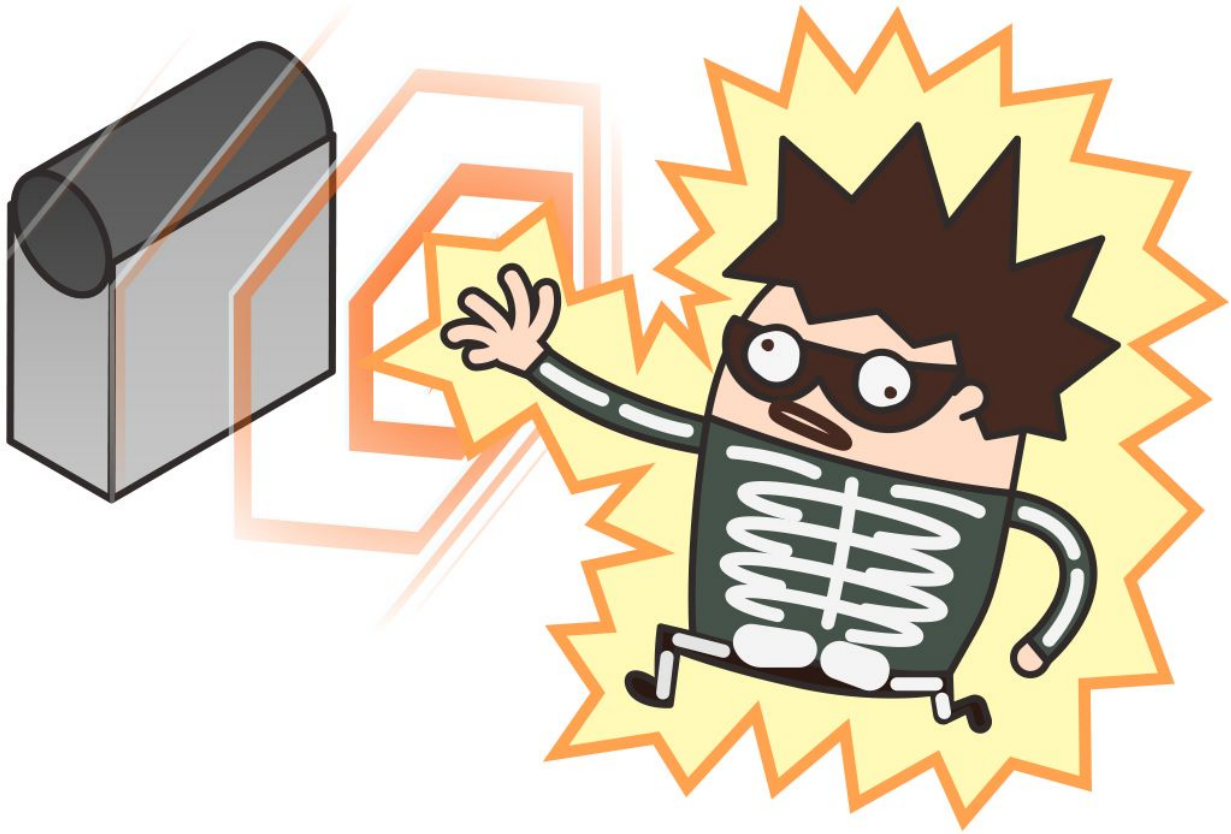

Figure: Malicious Access Will Be Blocked and Recorded

Deeper Connect employs triple encryption technology to guarantee the security of storage devices. Even if the hardware device is lost, nobody can crack into the data stored on the device.

Deeper Connect's triple encryption technology includes:

1. **Block Device Encryption**. If the storage device is lost, crackers could read critical files by analyzing the data on the block device. Each block on Deeper Connect is encrypted with AES-CBC [20] (see Figure 12), making it very difficult to crack the data because crackers can only access encrypted data.

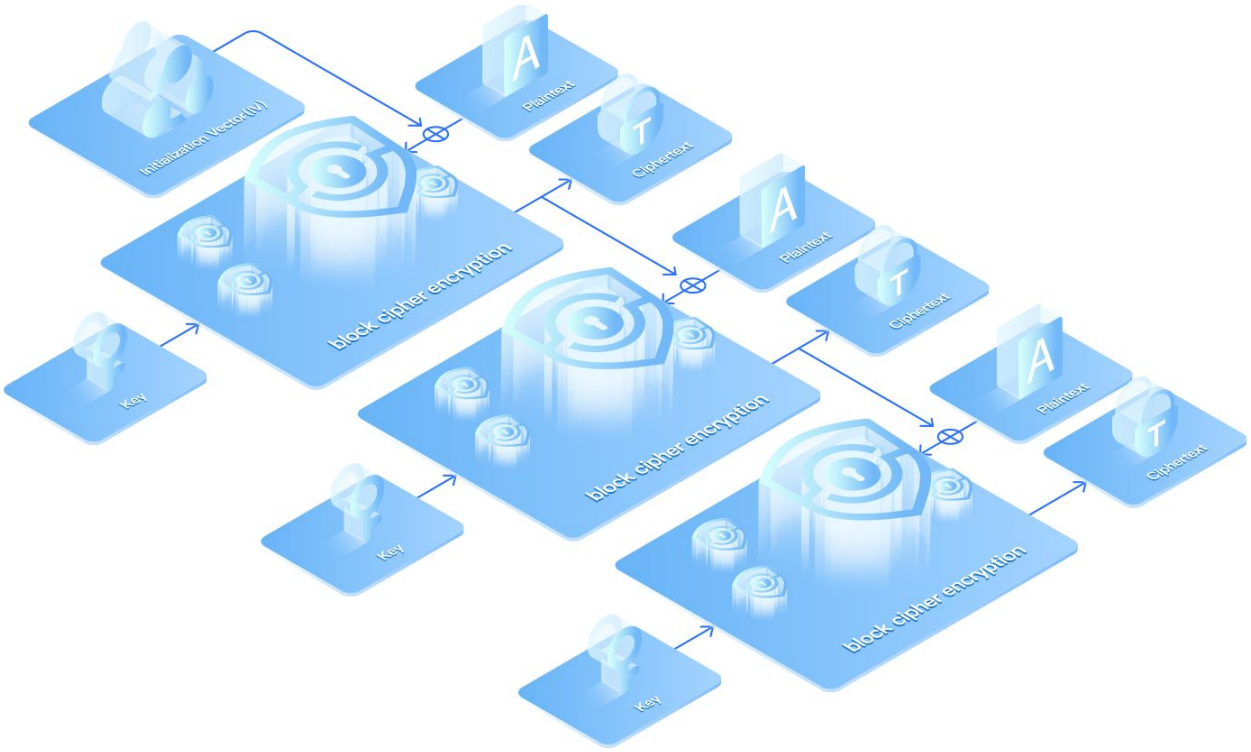

Figure: All Disk Data on Deeper Connect Is Encrypted with AES-CBC

2. **File System Encryption**. Simply employing block device encryption cannot ensure the security of the device. In order to further protect our storage media, Deeper Connect scrambled the key data structure of the general file system (see Figure 13). This is how DeeperFS is implemented, a unique file system of Deeper's own design. Due to the strict confidentiality of the data structure of DeeperFS, crackers cannot retrieve any information from the block device related to the structure of the file system, and thus

cannot access any critical file stored in the file system.

Figure: Encrypted File System Confounds Crackers

3. **File Encryption**. All critical files stored in the Deeper Connect file system have to be encrypted by AES-CBC. The decryption key for all files is only available within the compiled program code, meaning only the Deeper program can access the plain text information if needed (see Figure 14).

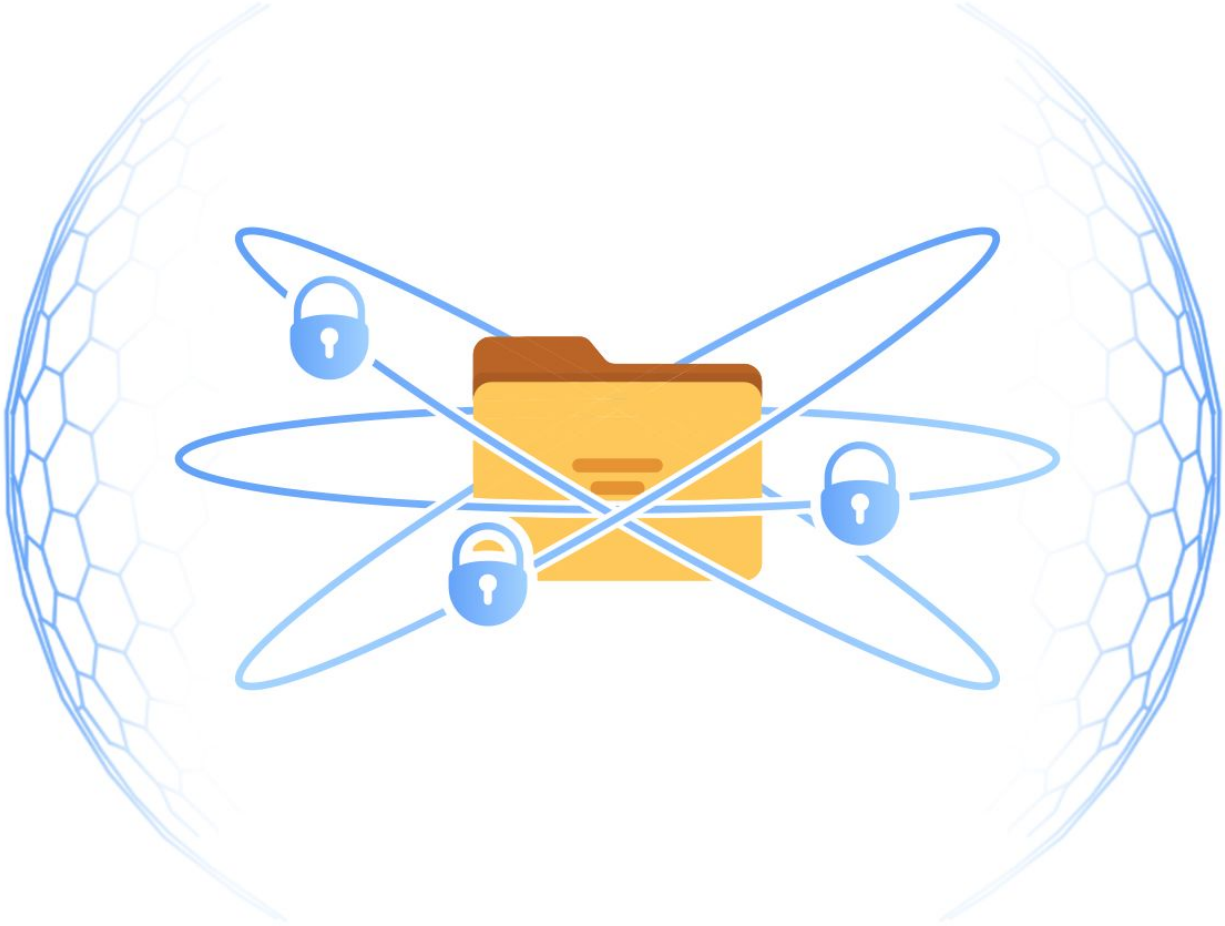

Figure: Triple Encryption Technology Guarantees Deeper Connect Data Security

Mining Rig with Network Security

On May 28, 2018, the “Packet of Death” was discovered in Ethereum (CVE-2018–12018) [55], where the attacker could freeze geth nodes by sending a death packet. Geth is the official client of Ethereum, extremely important for the Ethereum project: about 70% of nodes running geth contain key nodes for public exchanges and mining pools. With this bug, an attacker could torn down Ethereum and unleashed an earthquake on the Ethereum market.

After providing network sharing services, Deeper Connects will become e2 chain mining rigs as well. Currently, the security issues of mining rigs have been overlooked. However, if a cracker targets mining software bugs or mining hardware weaknesses, such an attack would naturally have a significant impact on the value of the corresponding cryptocurrency. All of Deeper’s products inherit network security genes and all of them are meticulously designed and fully tested. Deeper security devices running AtomOS will be the safest mining rigs in the world, maximally protecting the e2 chain and the interests of all its miners.

Figure: Deeper Connect with Its Inherited Network Security Genes Provides Addi- tional Protection for the e2 Chain

Operating system

AtomOS is a network operating system custom-built for deep security. It is also the world's first lock-free network operating system. The advanced design of AtomOS is the foundation of the reliability, efficiency, and security of the entire system. We next briefly introduce three aspects of AtomOS: packet I/O, packet scheduling, and deep packet inspection.

Packet I/O

Packet I/O falls into the I/O layer of AtomOS. It is one of the key technologies that determines user data flow latency and bandwidth throughput. Traditional operating systems use a kernel network stack to transmit and receive data. The main disadvantages of this approach are high latency and low throughput. After traversing the network to a network device, the packet encounters a series of intermediate processing hurdles such as the network interface card, network device driver, kernel network stack, and socket before undergoing final processing (see Figure xxx). In addition, this approach can incur frequent hardware interrupts and context switches between user space and kernel, further increasing data latency and reducing throughput.

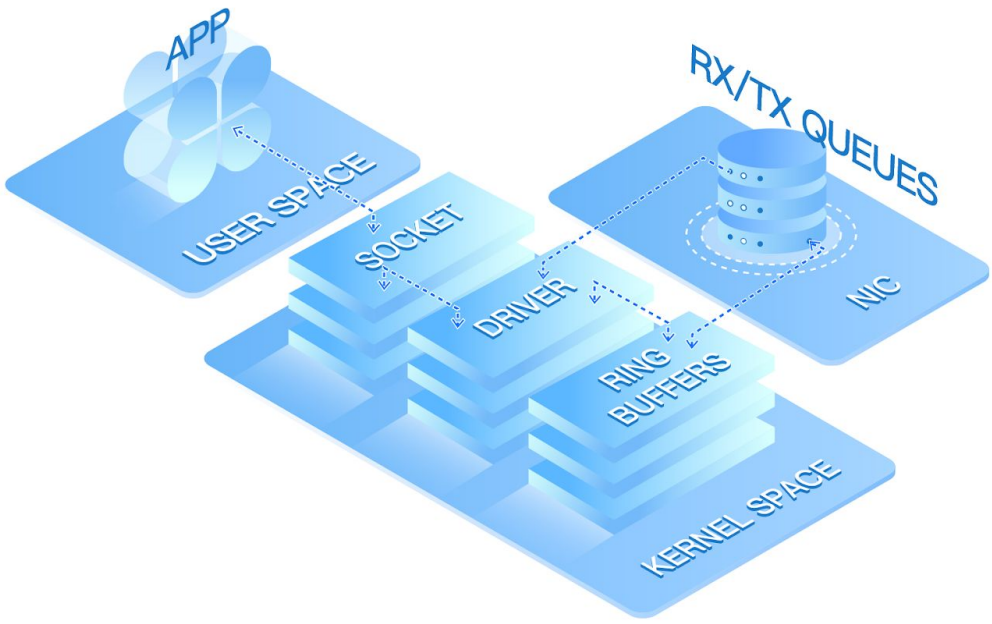

Figure: Traditional Operating System Data Transceiver

AtomOS employs zero-copy technology to access packets directly from network devices (see Figure xxx). This technology not only bypasses the cumbersome Linux kernel network stack, but also avoids frequent context switches and hardware interrupts. It greatly reduces data packet latency and increases throughput. AtomOS implements zero-copy technology with DPDK [xx], designed by Intel, which is reported to improve the throughput by tenfold [xx] compared to traditional kernel network I/O.

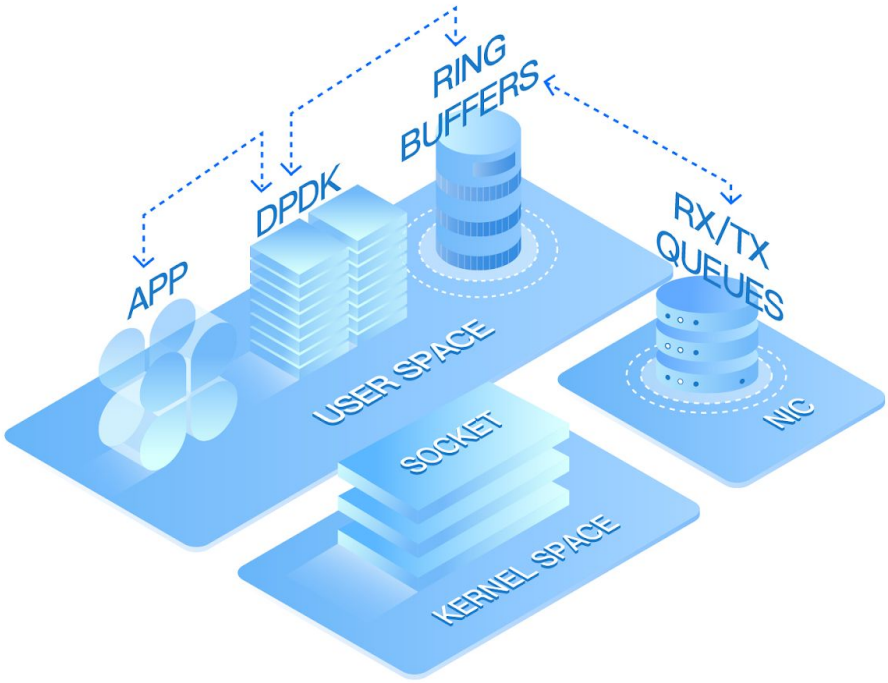

Figure: DPDK Data Transceiver

Packet Scheduling

High-performance network operating systems are facing big challenges from several perspectives. We first illustrate the challenges and then describe how AtomOS addresses them.

Challenges of Network OS

Performance and Scalability. As the size of CPU transistors decreases, Dennard's scaling law [xx] gradually breaks down. Reduced transistor size increases static power consumption and detonates serious thermal energy conversion. In addition, the accumulated heat between transistors is considerable, making CPU cooling an urgent issue. Simply increasing CPU frequency is no longer feasible due to the cooling issue. Therefore, major chip manufacturers have sensibly halted research on high-frequency chips. Instead, they have turned to low-frequency multi-core architecture. Cavium, a well-known processor manufacturer, launched a 48-core network processor back in 2012 [xx]. AMD released a 128-thread multi-core processor in 2019 [xx].

The development of multi-core processors also brings challenges for the design of network operating systems. Traditional network operating systems are usually based on vxWorks, FreeBSD, Linux or other classic operating systems. VxWorks was designed as a single-core embedded real-time operating system and has been phased out by network device vendors in the last decade. Both Linux and FreeBSD are derived from UNIX, whereas UNIX was originally

designed for control systems rather than data forwarding systems. The inherited design flaws of these classic operating systems make it difficult to enjoy the benefits of multi-core and even many-core processors.

Availability. Network operating systems are typically deployed at the boundaries of an assortment of network devices, meaning that if one network device is down, all connected devices in the network that rely on that device will also fail. Therefore, customers generally have extremely high demands for network device availability. In general, the availability of network equipment is required to reach 99.999%, that is, only five minutes of downtime per year is acceptable. Currently, network devices (especially network security devices) have to handle more traffic throughput and more features, making it increasingly challenging to maintain high availability.

Packet Order. When users access the Internet, dozens of network devices might be involved. If these devices do not maintain packet order, the data packets might be delivered in a completely random order. Packet disorder triggers the congestion control algorithm [xx] of the TCP protocol to reduce the size of the TCP transmission window, thereby seriously reducing the throughput of the data stream and affecting user experience. As mentioned above, multi-core and even many-core processors are now mainstream. Although multi-core processors can process data packets in parallel, serious out-of-order issues might occur without proper consideration. Harnessing the potential of multi-core processors while maintaining packet order has become a hard nut for network operating systems to crack.

Currently, all operating systems have to employ locks [xx] to solve these issues. However, lock design has in turn become an issue in network operating systems. If the granularity of the lock is too coarse, these big locks will become the bottleneck of the entire system for processors with more and more cores. On the other hand, if the granularity is too small, it might lead to deadlocks and race condition problems, even though operating system performance may improve. If not handled properly, these problems will significantly impact system stability.

HIPE

AtomOS implements the world's first lock-free network operating system with Deeper's unique HIPE data structure. All network operating system challenges mentioned above can be solved on an HIPE-based structure; it embodies the components of our design philosophy: simple, efficient and under control.

In order to satisfy the general needs of network systems and solve the issues of traditional operating systems, AtomOS employs the HIPE data structure to handle the global scheduling of shared resources in the network operating system. It ensures system correctness while taking full advantages of the benefits of multi-core performance.

In HIPE, various shared resources of the operating system are categorized into N groups. Large shared resources may span across multiple groups, and small shared resources belong to a single group (see Figure xx below).

Figure: Shared Resources Categorized into N Groups

Access to each resource group is triggered by events. Each event that needs to access a shared resource is placed into the lock-free queue for the corresponding resource group. When an event is popped from a queue, a CPU core is automatically assigned to process it. Since HIPE retains all events in the corresponding lock-free queue of each resource group, they must be processed sequentially without racing, thereby protecting shared resources.

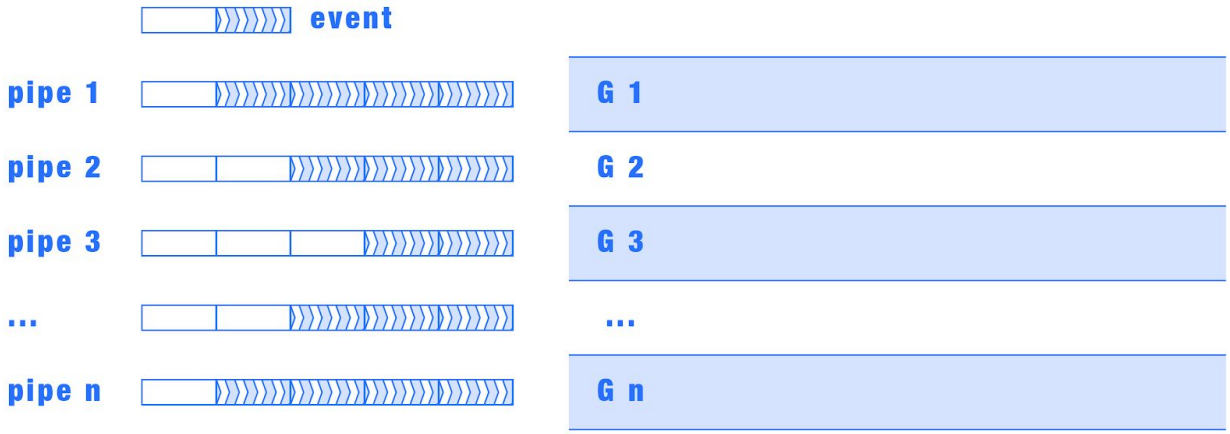

Figure: Access to Each Resource Group is Triggered by an Event in a Lock-free Queue

Since the number of resource groups in the system is much larger than the number of CPU cores, a continuous stream of data is available for each CPU to constantly process, making the performance of the entire system scalable with the number of CPU cores.

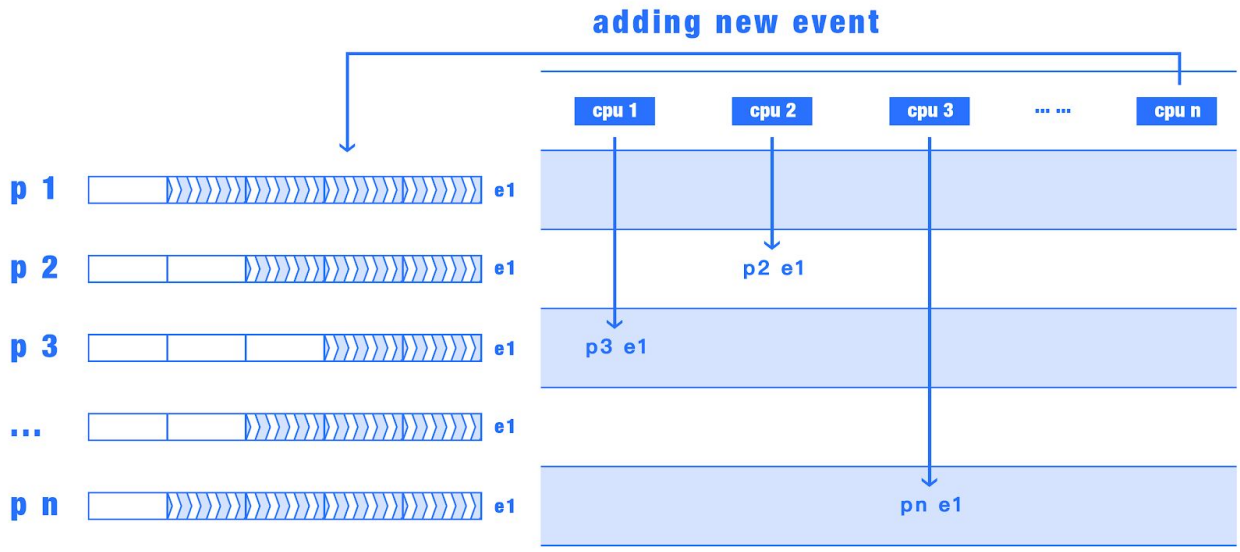

Figure: Resource Groups Processed in Parallel by CPUs

The lock-free design not only makes packet processing highly scalable, but also avoids the various race condition problems that spawn like flies when processes run in parallel. Moreover, since data packets sequentially traverse the HIPE pipeline, it guarantees that packet order in a particular data flow after the processing of AtomOS is consistent with its original order when receiving.

Deep Packet Inspection

Deep packet inspection is key for ensuring data flow under comprehensive protection. AtomOS provides connection security for each layer in the OSI model (see Table xx), which provides Deeper Connect with a complete set of network security functions.

7. Application layer	Application identification, malicious data flow detection
6. Presentation layer	Data encryption and decryption to prevent replay attacks
5. Session layer	Protocol session layer checking such as HTTP/SIP
4. Transport layer	Strict status check to prevent flood attacks
3. Network layer	Fragmentation attack protection, IP spoofing protection
2. Data link layer	ARP spoofing protection
1. Physical layer	Retaining connection during power failure

Table: OSI 7-Layer Protection in Depth

Nowadays the focus of network security has shifted from low layer to higher layer protocols. In addition to the various protections for network layers 1–3, AtomOS also implements the following advanced firewall functions for layers 4–7.

Strict TCP state checking: for each TCP connection, AtomOS keeps track of its state in the session table, and only the packets that strictly satisfy the TCP state machine will be forwarded. This prevents possible TCP masquerading and hijacking. At the same time, the authoritative NSS Labs firewall test cases in the industry was referenced during implementation to ensure containment of the various known TCP evasion methods.

Application identification and flow control: AtomOS integrates an application identification engine that is reliable, efficient, and scalable. It identifies common network traffic and performs flow control or smart routing to optimize the user experience for key applications. Also, it guarantees smooth tunnel service without consuming excessive local resources.

URL filtering: AtomOS can automatically filter malicious websites (including malware downloads, phishing websites, etc.) to provide a secure Internet environment. Users can also enable parental control to grade Internet content and assign proper access levels to each family member.

Network Address and Port Translation (NAPT): By default, AtomOS avoids network address and port translation for internal flows, to make it cable-live zero-configuration internet access. However, in some situations, AtomOS can utilize the symmetric mode of NAPT to further hide internal network structure if nec- essary.

Networking

Everything from the WP about routing, discovery, tunnelling, etc.

In addition to the feature of deeper packet inspection described in Section xxx, Deeper also independently designed Trident protocol, adaptive tunneling, intelligent routing, link layer tunneling, and tunnel layer congestion control. These technologies provide the deepest packet inspection and the best user experience.

Trident Protocol

The goal of Deeper's tunneling technology (implemented by the Trident protocol) is to circumvent network censorship. For various reasons, certain governments worldwide are now more frequently conducting deep inspection and filtering of user network traffic [33]. Network censorship relies on firewalls or offline traffic analysis devices deployed at the boundary of the core networks. Therefore, in order to introduce the Trident proto- col's bypassing feature, let us review the functionality of firewalls. Currently, firewall modes have been evolving from the basic

port based access control list to advanced con- tent based application identification. The advanced mode can be implemented in the following ways. The first four approaches belong to the passive identification method and the last one is proactive. Some firewalls can employ multiple approaches to identify applications of user data streams. Further, some artificial intelligence approaches such as Bayes' theorem [62] or decision tree [60] could be employed to perform application identification.

Basic Port Filtering. Basic port filtering refers to the application identification approach that is based on the destination port. The Internet Assigned Numbers Authority (IANA) [31] is the organization that allocates network ports and their corresponding network applications. As of now, almost all ports from 0 to 1024 have been allocated [43]. Firewalls are able to obtain a basic idea of user applications simply based on the network ports. For example, the destination port commonly used by the NFS protocol is 2049. Even without a clear content pattern, firewalls are still able to identify the application based on the specific destination port.

Content Identification. Content identification refers to the application identification approach that is based on the content of data streams. Since network applications have to follow the predefined network protocol, data streams tend to have a distinct content pattern. For example, the commands commonly used by HTTP (GET/POST, etc.) always appear as the first packet after TCP handshake. Also, the first line of data always ends with HTTP/X.X (the HTTP version used). Firewalls are able to identify HTTP applications happening on a particular destination port based on this pattern. Similarly, all standard protocols have an identifiable content pattern. For some non-standard protocols, content patterns might be changed due to protocol version upgrades, so firewalls have to regularly upgrade their content pattern databases as well to accommodate these changes.

Packet Length Identification. Packet length identification refers to the application identification approach based on packet length order or packet length distribution in data streams. This ap- proach works very well especially when no clear content pattern is available for data streams. The packet length traversed between client and server generally follows some pattern in the negotiation phase of a network protocol. If a network protocol specifies during the negotiation phase that the client has to send a TCP packet with a payload length of 60 bytes as a request, the server has to send a 40-byte packet as a reply followed by another 20–30 byte packet. In this case, the network protocol has a clear pattern in terms of packet length, which can be easily identified by a firewall. In order to evade packet length identification, ap- plications need to scramble or encrypt data packets to hide the pattern of packet length.

Packet Interval Identification. Packet interval identification refers to the application identification approach based on periodic keepalive packets specified in a network protocol. In the tunnel- ing protocol, the server and the client need to periodically send keepalive packets in order to monitor the availability of the tunnel. Keepalive packets generally are sent at a fixed interval and their size is fairly small. Non-standard tunnel- ing protocols still maintain this

pattern. As a result, firewalls used for network censorship can identify and block tunneling applications based on this pattern.

Active Detection Identification. Active detection identification means that the firewall acts as a middleman to modify data packet content between client and server, and identify the application according to the data packet content returned from the server. For example, IRC control channels are typically utilized by malware [58]. Even though they conform to the standard IRC protocol (a network chat protocol specified by IETF), they do not support the simple mutation of commonly used IRC commands. Based on this pattern, firewalls can proactively send requests and analyze the server reply to distinguish whether the network application is normal chat software or malware. This approach enables firewalls to monitor the content from data flows but also proactively modify or send data packets for application identification.

Targeting all of the above identification approaches, Trident protocol combines two tunnel modes to prevent any firewall identification attempts: protocol obfuscation mode and protocol camouflage mode. Since firewalls are unable to identify any traffic pattern in protocol obfuscation mode, internet censorship is not possible. However, for systems with whitelist, all unidentifiable applications are blocked as well. In this case, Trident protocol will automatically switch to protocol camouflage mode to circumvent internet censorship.

Protocol obfuscation mode includes the following key points:

- Random port
    - Randomly negotiate the data session port
- Encrypted content
    - All packet contents are encrypted
    - Ensure that content features cannot be expressed in regular expressions
- Obfuscation of packet length
    - All packet lengths are randomized
- No periodic keepalive data packets
    - Data packet piggybacks keepalive packet
    - No separate keepalive data packets exist
- Prevent active detection
    - Servers refuse to respond to any packets that do not follow protocol specifications

Protocol camouflage mode. There are two camouflage modes available:

- HTTP protocol: The tunneling protocol is completely encapsulated in an “HTTP GET” and an “HTTP POST” message body. The “GET Response” command is used to receive downstream data, and the POST message body is used to send upstream data. Since the port is negotiated by client and server in advance, no specific string name pattern is available in HTTP fields.
- TLS protocol: In this mode, the session ticket function of TLS 1.2 is used. The tun- nel traffic is like a standard HTTPS connection using the negotiated session ticket. Since

there is no negotiation phase, the firewall cannot decrypt or encrypt as a middleman. AtomOS will also use encryption and anti-identification mechanisms similar to the protocol obfuscation mode described above.

Another common issue with P2P networks is NAT [52] traversal. NAT is a common function of network devices in an IPv4 network environment. Network devices are typically configured with private IP addresses in LAN. However, in order to transmit packets out to the Internet, the destination IP address and source IP address of the packet must be translated to public IP addresses. In order to resolve this contradiction, the network device serving as a gateway can use NAT to convert the private IPv4 address into the public IP address of the gateway when the data packets are traveling from the LAN to the Internet. This approach not only solves the limitation issue of IPv4 addresses, but also satisfies the requirement from organizations to hide internal network structure and isolate external networks. In practice, Deeper Connect might sit behind the NAT device of service providers and assign it a private IP address. However, that would render Deeper Connect unable to receive connection requests from Internet devices. We use the following techniques to solve this problem:

- If the receiver side of the connection has a private IP address and the sender has a public IP address, the receiver initiates connection requests in reverse;
- If both sides use private IP addresses, NAT type identification is further required to determine the proper way to initiate the connection request. AtomOS im- plements a protocol similar to the STUN protocol (RFC3489 [61]). The network device is able to identify the NAT type and publish it along with other information about the node during the initial stage of network registration. The eventuality of both network devices using Symmetric NAT or Port Restricted Cone NAT can be avoided when setting up the connection. For the other NAT types (Cone NAT or Restricted Cone NAT), connection setup should provide a solution.

Adaptive Tunneling

Deeper Connect uses an efficient, flexible, and adaptive proprietary tunneling protocol rather than a standard one such as IPSEC. In the process of designing and imple- menting adaptive tunneling technology, we have borrowed extensively from various industry-approved WAN acceleration technologies [73]. Given the high latency, high packet loss rate and out-of-order issues of multinational Internet, we improved these technologies in the data tunnel layer, which effectively maximizes bandwidth utilization and significantly improves the user’s online experience.

Adaptive Data Compression and Merging

With adaptive tunneling technology, Deeper Connect can determine if packets in the data stream are compressible and decide whether to perform compression. For instance, the most common HTTP protocol is composed of mainly Latin charac- ters, which can be compressed to save approximately 70% bandwidth and thereby greatly improve transmission efficiency.

Meanwhile, given the fact that MP4 and other formats commonly used in video and audio traffic (or networks protocols such as HTTPS/SFTP which uses SSL and TLS encryption) have already ap- proached the theoretical limit of information entropy [64], additional compres- sion would only increase CPU consumption without saving bandwidth, resulting in compression processing and in turn transmission rate reduction. Therefore, adaptive tunneling needs to identify and process accordingly based on content for both CPU and bandwidth efficiency.

Through adaptive tunneling technology, Deeper Connect can also improve trans- mission efficiency by combining small data packets. Many network protocols have a large amount of control packets with little or no data in the payload. Taking a 30KB HTTP transport stream as an example, even if the client's protocol stack optimizes TCP ACK for every two packets, 40% of packets are still less than 100 bytes. Such a large proportion of packets containing a very small amount of data causes considerable transmission efficiency lag. For optimal transmission efficiency, adaptive tunneling technology can combine or compress and transmit data packets from multiple data streams without affecting the TCP connection latency (see Figure xx).

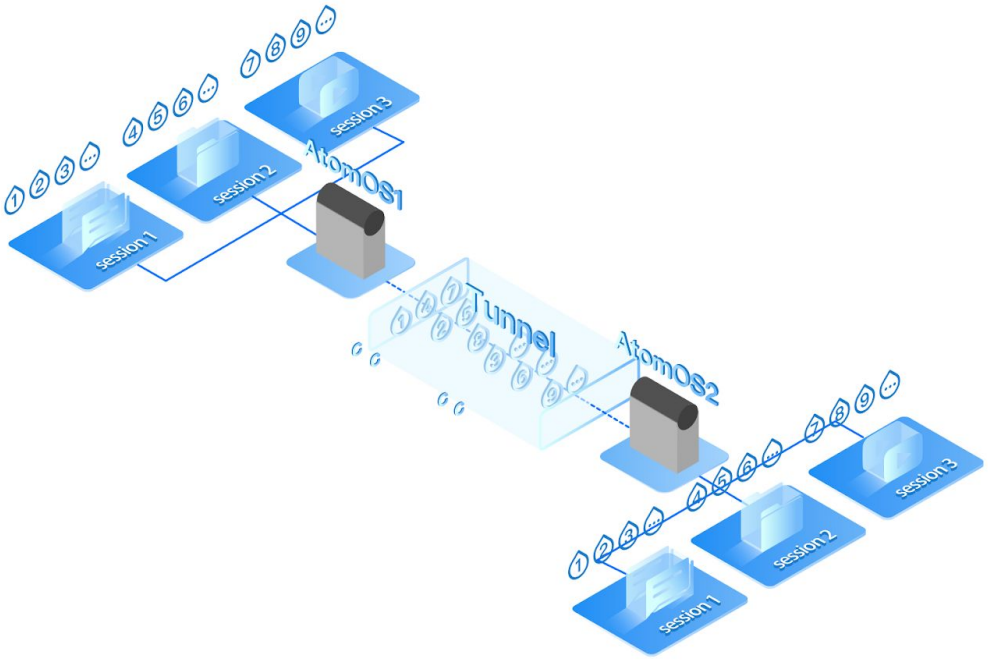

Figure: Automated Packet Consolidation, Compressed Transfer Schematic

Application-Based Traffic Control

Application-based traffic control functions according to the application type of the data stream, to ensure that latency-sensitive or volume-sensitive applications enjoy a higher QoS level. In a home network, bandwidth is often limited. When multiple applications are used simultaneously, the demand for bandwidth is often much larger than what is available. To address this allocation issue, adaptive tun- neling can automatically determine application type according to the user data stream and grant the corresponding QoS level. For example, web browsing or email

downloads should be classified as latency-sensitive, whereas applications such as file downloads are not. Adaptive tunneling first automatically estimates the network tunnel's actual bandwidth and its bandwidth requirements. If de- mand exceeds supply, adaptive tunneling will control bandwidth usage based on the application's QoS level. Lower level applications will be temporarily buffered in a limited packet queue. If the packet queue is full, overflow packets will be discarded. Although general application use may be affected due to increased latency and packet loss, overall user experience is improved significantly.

Intelligent Routing

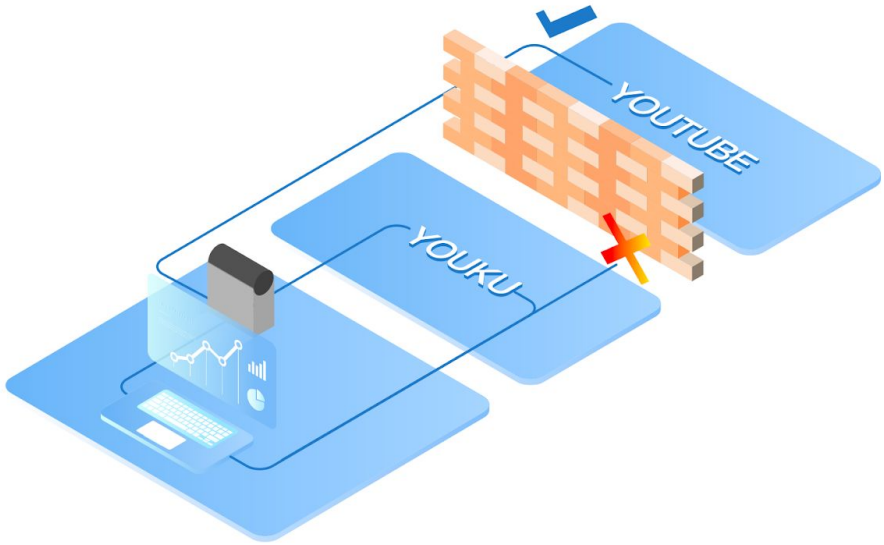

Figure: Intelligent Routing

Intelligent routing refers to automatic configuration of network routing based on data stream characteristics, and whether to transmit through a tunnel. We offer two modes, a privacy protection mode and a network circumvent mode. The default mode is network circumvent mode.

Privacy protection mode: In this mode, all data flows related to tracing online browsing will be processed through the tunnel depending on the anonymity level set by the user.

Network circumvent mode: In this mode, all online data flows will be processed over the tunnel depending on whether or not the website database shows if it is blocked in the local area.

Intelligent routing provides users with the following benefits:

- **Monetary Savings**. Network tunnels are established by two or more Deeper Connects. When one Deeper Connect tries to connect with another one to establish a tunnel, cryptocurrency payment (calculated according to bandwidth and traffic volume) is required through the secure shared network platform. The implementation of the network sharing platform will be elaborated in Section 3.5. Obviously, tunnel- ing services cannot be offered free of charge. Intelligent routing automatically determines whether to transmit through the tunnel according to the attributes of the data stream. This approach not only reduces the amount of tunnel usage, but also avoids latency caused by tunneling, providing a better online experience without incurring additional expenses.
- **Anonymity Service**. Anonymity service refers to hiding the user's IP address to sidestep tracking. Since the network tunnel is end-to-end encrypted, the data stream transmitted through it will leave no trace. We will set levels according to user access object visibility, and based on user settings, decide whether to perform encapsulation on the corresponding data stream. Highly visible user data streams such as web page visits are at the highest level of anonymity service. For this level of the user data stream, encapsulation is mandatory. The less publicly available user data streams such as P2P downloads belong to the second highest level of anonymous services. For this level, encapsulation is an optional setting to reduce user costs. Not only that, users can also choose a multi-hop routing mode for more rigorous anonymity services. In a multi-hop routing environment, the network tunnel will be established by several Deeper Connects instead of the usual two. The advantage of this is that Deeper Connect, as an intermediate node, cannot peek at the content because it cannot decrypt the user data stream. The last Deeper Connect node can decrypt the user data stream but cannot know the source. Therefore, the more Deeper Connects nodes in the route, the more difficult it is to track user activities.

Link Layer Tunneling

AtomOS is the world's first zero-configuration OS that can implement intelligent routing and tunnel encapsulation in virtual wire mode. All network devices currently on the market that implement the tunnel function work in routing mode. That is, the user needs to have certain network technology as well as working knowledge of IP address planning and tunnel protocol configuration in order to correctly establish the tunnel. It also requires a certain amount of routing knowledge to forward the required traffic to the tunnel for proper encapsulation and decapsulation. AtomOS completely changes this, for no professional know-how is required of Deeper Connect users. After the user connects the AtomOS device to the home router uplink, AtomOS will enter the learning phase. It does not affect the forwarding of traffic, and automatically determines the direction of its connection according to the statistical rules of the IP addresses that appear on the two ports. There are hundreds of millions of nodes on the Internet, while the number of local IP addresses are relatively small and fixed. So after briefly analyzing traffic, we can tell which is the uplink port and which the downlink. AtomOS will proceed to learn the uplink IP/MAC address, DNS server and other information for future tunnel negotiation and encapsulation.

We believe that the smart home gateway itself is a product with very low user operation frequency. There is no need for users to be aware of its existence most of the time, and little configuration is required to alter functions. Particularly in combination with our unique intelligent routing technology, user privacy and network transmission requirements are fulfilled at the lowest cost with no learning curve at all.

Tunnel Congestion Control

One of the key use cases of the Deeper Network is to provide users with network anonymity, which protects their privacy and enables them open access to Internet con- tent without being censored or blocked. In the anonymity service (as shown in Figure 23), the user transmits data through the secure AtomOS tunnel between the Deeper nodes, so that the accessed Internet service cannot track the user's private data (e.g., IP address, location). At the same time, since data packets in the AtomOS tunnel are strictly encrypted, censorship firewalls are effectively blinded and unable to identify the Internet content being accessed by the user.

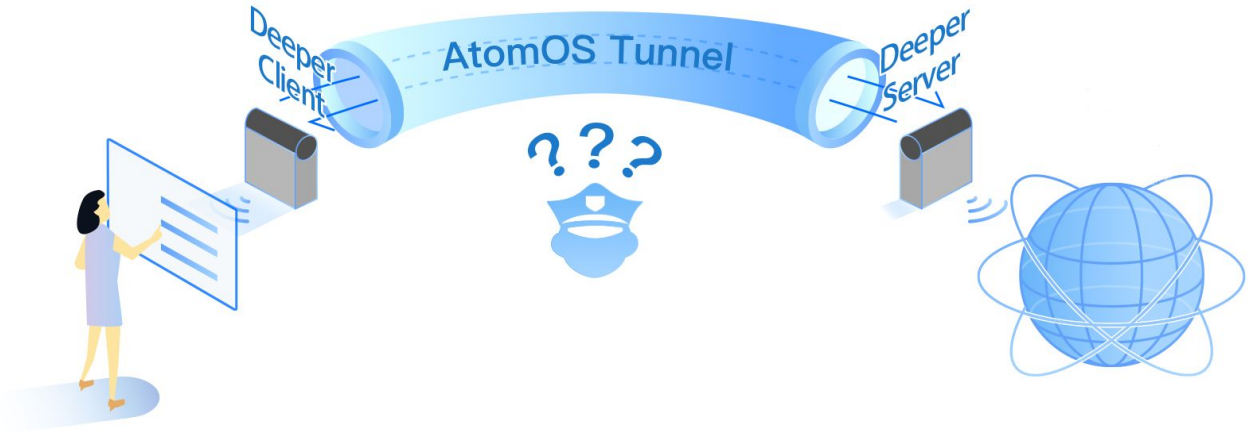

Figure: Secure Shared Service (SSS)

Through the combination of Deeper's unique network security and blockchain tech- nologies, SSS effectively ensures the security and stability of the Deeper Network's anonymity services. However, the efficiency of data transmission in the AtomOS tunnel remains an open question. With SSS, there are two major challenges to data transmission:

1. SSS is primarily intended to access Internet content in other countries or re- gions. Long-distance data transmission, large transmission delay, and high packet loss/out-of-order rate are problems associated with such international Internet access.
2. Although packets in the AtomOS tunnel are strictly encrypted and thus censor- ship firewalls cannot identify them, the firewalls may adopt a random packet drop policy (e.g., 1% random packet drop) for unrecognized data streams in order to downgrade their user experience.

To address the above challenges, Deeper is pioneering a connection-oriented, reliable transmission protocol at the tunnel layer. This is mainly to solve the problem of data transmission efficiency in SSS from the perspective of network congestion control. The complete set of congestion control solutions in the Deeper Network is called TBBR (Tunnel Bottleneck Bandwidth and Round-trip propagation time). It is composed of two core parts: 1) Deploying the new congestion control algorithm called BBR in the AtomOS tunnel, so that in case of high packet loss rate, the AtomOS tunnel can still maintain high transmission rate and low transmission latency; 2) Enabling fast packet loss detection and retransmission, so as to better adapt to high packet loss rate in SSS.

TBBR mainly focuses on improvements on the sender side. The receiver side need not make any changes. The sender does not rely on any additional feedback from the receiver. This is one of the important design principles of TBBR. It enables easier deployment of TBBR as no changes are required from the receiver side. More impor- tantly, in the high latency and high packet loss rate scenario of SSS, any additional feedback from the receiver side will undoubtedly increase network load, and in such an environment, no stable feedback can be guaranteed.

Traditional congestion control algorithms (such as CUBIC [26], TCP Vegas [7], 45
TCP Reno [54]) are usually based on packet loss events. Packet loss is treated as a signal of network congestion. These kinds of algorithms control data sending rate via a sending window. The window size W(t) at time t is controlled by the AIMD (Additive-Increase/Multiplicative-Decrease) algorithm:

$$W(t+1) = \begin{cases} W(t) + \alpha & \text{if no packets loss is detected} \\ W(t) * \beta & \text{otherwise} \end{cases}$$

Clearly, the AIMD algorithm tends to keep increasing window size (i.e., transmission rate) until packet loss is detected. Once packet loss is detected, the window size will experience a sharp drop. This leads to two main problems:

1. It is counterproductive to treat all packet loss events as signals of network conges- tion. In fact, packet loss can also be caused by network errors. In addition, when using SSS, censorship firewalls may also deliberately drop packets. According to the AIMD algorithm, when packet loss occurs, transmission rate is drastically re- duced. When packet loss rate reaches a certain level (e.g., 1% packet loss caused by censorship firewalls), the entire network transmission bogs down.
2. Since AIMD keeps increasing transmission rate until packet loss is detected, such a mechanism tends to fill up the entire network buffer (i.e., queue). The greater the number of packets waiting in the queue, the higher the queuing delay. Since memory prices are

becoming cheaper and cheaper in recent years, network buffer space is increasing accordingly, which leads to tremendous queuing delays.

It can be seen that traditional congestion control algorithms achieve neither optimal transmission rate nor optimal network latency.

Deeper deploys a new type of congestion control algorithm called TBBR at the Ato- mOS tunnel. TBBR was developed based on the BBR algorithm [10] combined with tunneling technologies. BBR was first introduced by Google and has been widely de- ployed in Google's WAN (Wide Area Network). Unlike traditional congestion control algorithms, TBBR/BBR no longer relies on packet loss events as signals of network congestion, but goes back to the essence of network congestion: The sender side is transmitting data faster than what network capacity can handle. In order to mea- sure current network capability, TBBR/BBR continuously measures two key metrics, namely, BtlBw (Bottleneck Bandwidth) and RTprop (Round-trip propagation time). If the network path were a water pipe, the bottleneck bandwidth BtlBw would be the min- imum diameter and the round-trip propagation time RTprop would be the length. The capacity of the entire network, i.e., BDP (Bandwidth Delay Product), is the product of the two:

$$BDP = BtlBW * RTprop$$

BDP can also be interpreted as the maximum amount of outstanding data that can be carried in the network without causing any queuing delay (i.e., without occupying any buffer space).

The main idea of TBBR/BBR is that when the data arrival rate at the network bottleneck equals BtlBw and the amount of inflight data in the network equals network capacity BDP, the network is operating at the optimal state of maximum throughput and minimum latency. TBBR/BBR controls transmission rate by measuring BtlBw and RTprop. It is worth noting that the capacity of the entire network is dynamically changing. Thus, TBBR/BBR must continuously measure BtlBw and RTprop to update transmission rate. In addition, BtlBw and RTprop cannot be measured at the same time. In order to measure BtlBw, one must fill up the network buffer to obtain maximum throughput; in order to measure RTprop, the network buffer must be as empty as possible (i.e., no queuing delay) to obtain minimum latency. To address this problem, TBBR/BBR measures the two metrics alternatively and estimates them by using the sampled values over a certain time window WR at time T :

$$\hat{BtlBw} = \max(r_t), \forall t \in [T - W_R, T]$$

$$\hat{RTprop} = \min(RTT_t), \forall t \in [T - W_R, T]$$

Where rt is the measured data transmission rate at time t, and RTTt is the measured round-trip time at time t.

TBBR/BBR possesses the following two properties:

1. At a certain packet loss rate, TBBR/BBR still maintains a stable transmission rate that is close to the network bandwidth.
2. While maintaining the maximum throughput, TBBR/BBR tends to not occupy the network buffer, and thus reduces queuing delay.

Google has deployed BBR on their Google.com and YouTube servers. BBR has success- fully reduced YouTube's median network transmission latency by 53%. In developing countries, this value is as high as 80% [xx].

Deeper has transplanted the successful experience of BBR into the application of SSS, and deployed TBBR, the world's first tunnel congestion control, into the AtomOS tunnel. With TBBR, we find that Deeper Connect effectively reduces international Internet access delay while still maintaining a stable network transmission rate when firewalls deliberately cause packet drops.

Figure xx compares the network throughput of the AtomOS tunnel with TBBR and that of the traditional tunnel IPSEC without congestion control under different packet loss rates. The experimental setup is 1 data stream, BtlBW = 100Mbps, and RTT = 100ms. The gray curve at the top represents ideal transmission rate, i.e., BtlBW * (1 - p), where p is packet loss rate. As we can see from the figure, a very small packet loss rate (0.01%) can cause the throughput of IPSEC to drop to only 30% bandwidth. As packet loss rate increases, IPSEC has a throughput of only 5% of remaining bandwidth, where the transmission is almost paused. In sharp contrast, the throughput of the AtomOS tunnel stays close to ideal throughput even at an extreme 5% packet loss rate. At 15% packet loss, the AtomOS tunnel still maintains 75% bandwidth. In SSS, assuming censorship firewalls randomly drop 1% of unrecognized packets, the throughput of the AtomOS tunnel would be virtually unaffected and would stay close to ideal throughput; while IPSEC would have a throughput of only 5% of remaining bandwidth.

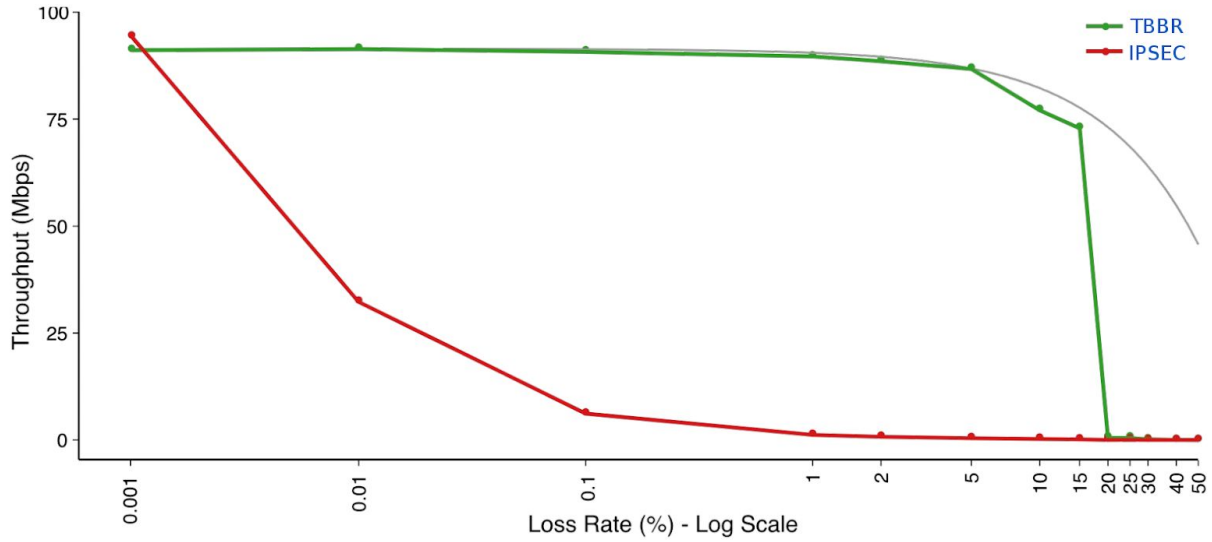

Figure: Network Throughput at Different Packet Loss Rates

Figure xx compares the network latency of the AtomOS tunnel and IPSEC at dif- ferent buffer sizes. The experimental setup is 8 data streams, BtlBW = 128kbps, and RTT = 40ms. The traditional tunnel IPSEC tends to occupy the entire network buffer space, which causes latency to increase linearly with buffer size. Even worse, if latency is larger than the network initial connection (SYN) timeout set by different operating systems, it will cause the connection to fail. In sharp contrast, the AtomOS tunnel always keeps latency to a minimum regardless of buffer size.

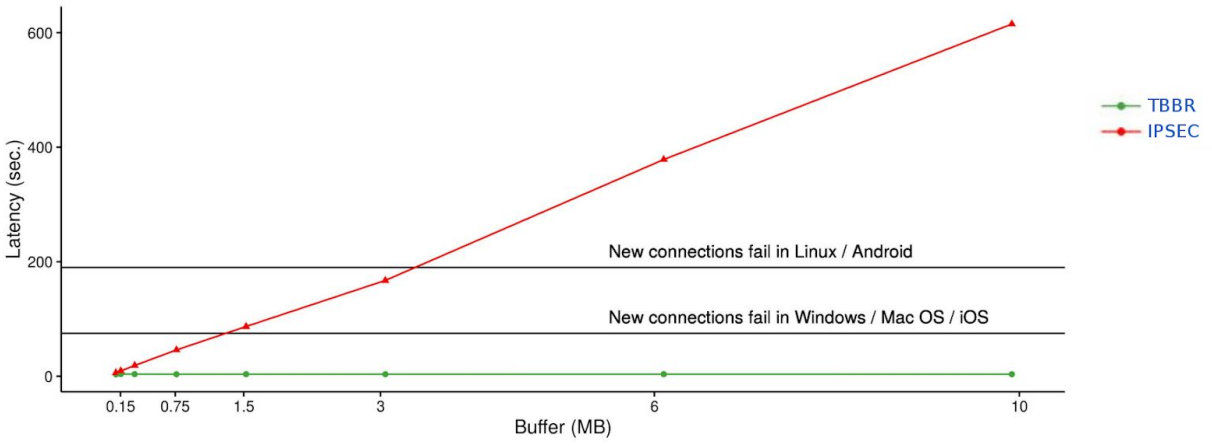

Figure: Network Latency for Different Buffer Sizes

On top of BBR, the AtomOS tunnel implements further optimizations for fast packet loss detection and retransmission.

Traditional TCP mainly handles packet loss in two ways:

1. If the acknowledgment (ACK) of a packet is not received within a certain time period, i.e., retransmission timeout (RTO), the packet is considered lost and re- transmission is triggered.
2. Instead of waiting for timeout, if three duplicate ACKs are received from the receiver, the sender also considers a packet as lost and triggers retransmission. This mechanism is called fast retransmission.

In TCP, when the receiver finds that some packets were skipped, it will send duplicate ACKs to remind the sender that some packets are still missing. There are two reasons a packet may be skipped: either it is lost or the packets arrived out of order, i.e., packets originally scheduled after a certain packet arrived at the receiver side first. When the sender receives a duplicate ACK, it cannot immediately determine which of the two scenarios occurred. Therefore, it is necessary to wait for further duplicate ACKs to determine that packet loss happened with a high probability. If packet loss is determined prematurely, it will lead to unnecessary retransmission that increases network load; on the other hand, if packet loss is determined too late, it will cause a slow response to packet loss events.

Today, a commonly used fast retransmission mechanism is based on three duplicate ACKs. It requires at least 4 data packets to be sent (i.e., the sending window size is at least 4) to observe three duplicate ACKs; otherwise, the sender can only rely on RTO timeout for retransmission. Therefore, the current fast retransmission mechanism works poorly or not at all in the following cases:

1. Studies [xx] have shown that from the perspective of the application layer, a TCP connection often needs to send a total of less than four data packets. In these cases, the current fast retransmission mechanism will never be triggered.
2. Network congestion may cause the sending window to shrink below 4, which also disables fast retransmission.
3. In cumulative ACK mode, the receiver may choose to delay sending ACKs to merge multiple ACKs into one in order to save bandwidth. In this case, even more data packets are needed to be able to trigger fast retransmission.

Algorithm 1 Algorithm for fast retransmission threshold τ in TBBR

Assume that the number of currently unacknowledged packets is k
if there are no more packets to send **then**
    $\tau = \max(\min(k-1, 3), 0)$
else
    $\tau = 3$

An effective fast retransmission mechanism should detect packet loss and trigger retransmission in time while reducing superfluous retransmissions. TBBR adopts a dynamic fast retransmission threshold algorithm. In a nutshell, if no more data packets can be sent (either due to sending window size limit or because the application layer has no more data to send),

the threshold of fast retransmission is dynamically adjusted according to the number of packets that have not yet been acknowledged; otherwise, a threshold of 3 is used.

Regarding retransmission timeout RTO, traditional TCP adopts an algorithm called exponential backoff, i.e., if a packet times out under the current RTO, the packet is retransmitted and the RTO is doubled. In extreme cases, if packet timeout happens n consecutive times, the RTO will explode to 2n times the original RTO, which greatly stalls transmission rate. TBBR uses a smoother RTO growth curve that sets RTO to 1.5 times the previous value per timeout.
Although the overall design of TBBR is focused on the sender side, we can still improve network transmission efficiency from the receiver side. There are two main approaches:

1. Adopt selective acknowledgement (SACK [xx]) at the receiver side. In contrast to the cumulative acknowledgment where the receiver only feeds back the minimum sequence number of the packets that have not been received yet, SACK allows the receiver to explicitly tell the sender which packets have been received and which have not. The sender can selectively retransmit only those packets that have not yet been received. In addition, if multiple data packets are lost in the current sending window, cumulative acknowledgement only informs the sender of one packet loss at a time, resulting in inefficiency. SACK can feed back all lost packets at once. Research shows that in high latency and high loss rate networks, SACK can greatly reduce the number of retransmitted packets and improve transmission efficiency.
2. Dynamically adjust the acknowledgment delay. As mentioned earlier, the receiver can choose to delay sending ACKs. While doing so makes better use of band- width, it also delays packet acknowledgement and holds back fast retransmission. Especially in a high delay and high packet loss environment, it is crucial that the receiver acknowledges every packet in time. Therefore, at the receiver side, acknowledgement delay can be adjusted dynamically according to the delay and packet loss conditions of the current network.

Blockchain

Overview

- Design goals
- Participants
- Consensus mechanism used (briefly)
- Programming capacity (what can and cannot be done)

Consensus mechanism

Overview

Deeper uses HotStuff as its state machine replication (SMR) framework. HotStuff is a novel framework that has some useful properties that make it especially suitable for a privacy-oriented and highly distributed network.

HotStuff works in the so-called “chaining” paradigm, which is inspired by blockchain and allows to greatly improve the throughput of the network. In the chaining paradigm, instead of explicitly voting for a proposal in round phases (i.e., propose, pre-commit, commit, etc), a phase is generalized and the vote on a proposal that references some parent is considered a vote for the next phase for the parent. E.g., a vote on a block is considered a prepare vote for the block itself, a pre-commit vote for its parent, and a commit vote for its grandparent, and a decide vote for the 3rd-generation ancestor. The block is executed only when its 3rd-generation descendant is successfully voted on.

In practice, HotStuff presents roughly the same latency as in BFT, while processing several blocks in the time it takes ordinary BFT to process one, which leads to higher throughput.

Additionally, HotStuff uses a star communication pattern (i.e., everyone communicates through the leader) and threshold signatures to ensure linear communication per block - the leader sends the block to validators, they produce partial signatures, and the leader simply reconstructs a threshold signature that serves as a proof of block validity. This allows to scale consensus to a large number of validators simultaneously.

Liveness and leader selection

Hotstuff in the basic form abstracts the liveness and leader selection parts of consensus. These functions have to be implemented separately in each particular instantiation of Hotstuff, depending on the network topology and validator dynamics.

In Deeper, both liveness and leader selection are performed through computation of VDFs, which is done by a separate consensus process called Pacemaker.

VDFs are similar to VRFs - cryptographic functions with unpredictable output, which also generate a proof that can be used to verify the correctness of VRF’s computation. VDFs, additionally, require a predetermined number of sequential steps to compute, which puts a lower bound on the time it takes to produce an output. At the same time, verifying a VDF is significantly faster than computing it.

In Deeper, the time-bound on the computation of the VDFs is used to ensure liveness, while the randomness of the output allows for leader selection that isn’t tamperable. The output is unpredictable and delayed, which ensures that the adversary can’t know the leader ahead of

time and corrupt them. At the same time, the output is deterministic and can't be changed by the adversary after the block containing the input is added to the chain.

Let
S_t - the threshold signature constructed by the leader from validator partial signatures,
$H(x)$ - a hash function,
$(y, p) = VDF(x, t)$ - the VDF with the input parameter x and time parameter t that produces the output y and the proof p,
T_b - time parameter that produces a lower bound equal to a single round time when passed to the VDF,
TW - the sum of all validator weights from PoS/PoC (see more in "Tokenomics")

When the leader reconstructs S^k_t for the round k, it is sent to the validators so that they join the proposal. Immediately, the Pacemaker of each of the validators (including the leader) starts computing $(y_k, p_k) = VDF(S_t, T_b)$. With this time parameter, the VDF should be computed by the validators after the round $k+1$ but before the end of voting in round $k+2$.

The leader for the round $(k+3)$ is selected as $H(y_k||1) \% TW$. Once the validator computes the VDF, they send a tuple $(y_k, p_k, H(y_k||1), 1)$ to the new leader, which is to be treated as a "new round" message.

If the leader is correct, they send the new proposal, as well as the leader proof tuple, to all validators and proceed with the round.

If the leader is faulty, reacquiring liveness is done depending on the timeout length.

1 round time out
The validators need to determine that the leader timed out - the natural way to do that is to wait until the VDF result (y_{k+1}, p_{k+1}) for the round $k+4$. If the Pacemaker computes the VDF for the round $k+4$, but the leader for round $k+3$ has not started voting, the validator signals timeout by computing the new round leader $H(y_k||2) \% TW$ and sending it the tuple $(y_k, p_k, H(y_k||2), 2, y_{k+1}, p_{k+1})$. $(y_k, p_k, H(y_k||2), 2)$ serves as a leader proof, while (y_{k+1}, p_{k+1}) is the timeout proof. Right after that, it starts computing $(y^t_k, p^t_k) = VDF(y_{k+1}, T_b)$.

At this point, if the new leader is correct, it generates the proposal and sends it to the validators together with the leader proof and timeout proof. Simultaneously, Pacemaker should finish computing (y_{k+2}, p_{k+2}) - this is used to select a leader for round $k+4$. The leader's and the validators' Pacemakers should have already started computing $(y^t_k, p^t_k) = VDF(y_{k+1}, T_b)$, which is used to select the leader for the round $k+5$. From there, the protocol proceeds as normal.

If the validator receives a timeout message for round $k+3$ but they have a joined block for round $k+3$, they should respond with that block. The validator that sent the timeout message and received back a block should proceed with the protocol as normal.

2+ round time out

If the Pacemaker times out again (i.e., computes $(y_{k+2},\ p_{k+2})$), the validator repeats the same process with an increased increment, i.e. sends $(y_k,\ p_k,\ H(y_k||3),\ 3,\ y_{k+2},\ p_{k+2})$ to the new leader. It also starts computing $(y^{2t}{}_k,\ p^{2t}{}_k) = VDF(y_{k+2},\ T_b).$

From here on, the validator continues to retry leaders, until it gets a block with a QC from a for round $(k+3)$, from the initial leader or one of the timeout leaders. They also continue to compute VDFs as follows: for the n-th retry they start computing $(y^{(n-1)t}{}_k, p^{(n-1)t}{}_k) = VDF(y^{(n-2)t}{}_k,\ T_b)$. If the retry succeeds, $(y^{(n-2)t}{}_k,\ p^{(n-2)t}{}_k)$ and $(y^{(n-1)t}{}_k, p^{(n-1)t}{}_k)$ are used for rounds $(k+4)$ and $(k+5)$, respectively.

Tokenomics

Overview

- What are the main uses of the token
- Token total supply and method of emission
- What mechanisms is token involved in

Staking system

- What goes into staking
- Rewards for block validation
- Slashing and/or freezing conditions if they exist
- The tables about supply growth go here

Proof of credit

The e2 network consists of two layers. The first layer contains about (e.g.) 100 validators that generate new blocks constantly. The application layer consists of millions to billions of deeper network devices. Proof of credit allows deeper network devices to mine new tokens by sharing their bandwidths. Each device will be associated with an account i.e. public address. The more bandwidth a device shares, the more credit score will be earned by the account. Each device will delegate its voting power (a function determined by its credit score) to a validator. The consensus of the e2 chain will be reached if more than ⅔ of total voting power of validators agree on a new block. After the new block is mined, the accounts will be rewarded proportional

to their credit scores. As in any credit score systems of modern society, each account's credit score is capped by some maximum value.

Micropayment and credit score update

There are two roles of deeper network devices. When a device provides bandwidth to other devices, we call it a server device. When a device served by a server device is called a client device. For each MB of data the server device provides to the client, the client will pay the server account with X tokens. This is so-called micropayments. If the client doesn't pay the micropayment, the server can stop providing bandwidth. We allow the server to choose the micropayment period, i.e. stop service after serving Y MB of data without micropayment. The micropayments happen in the application layer which is off-chain. A device can accumulate multiple micropayments and submit the transaction history to the validators. The credit score of the associated account will be updated.

Proof of credit and its security

Sybil attack prevention is a key security consideration in public blockchain. There are many different approaches: proof of work, proof of stake, delegate proof of stake etc. Bitcoin and Ethereum 1.0 use proof of work where a validator solves a difficult cryptographic puzzle in order to create a new block. Cosmos and Algorand use proof of stake where an elected validator will vote for a new block and the voting power is proportional to the total amount of tokens it staked. Deeper network uses a similar approach as Proof of Stake. However, the voting power does not just depend on the staked tokens but also on the credit score of an account. Thus, e2 chain is actually a mix of Proof of Stake and Proof of Credit. The security of Proof of Stake is well-studied. Hence, our major concern is the security of proof of credit.

Network model

The topology of the application layer is a big graph that contains millions to billions of nodes (devices). Let's assume the total number of nodes is N and the number of malicious nodes are X. During each epoch (a period of time, e.g. one week), we consider the random graph is fixed.

API

We define several functions here:

Fn randomize_graph() -> Graph<V,E> // smart contract in layer 1 that return a randomized graph before the current epoch start

Fn nbr<V:Node>(n: V) -> Vec<V> // given a node, return a list of neighbor peers that this node can connect to during current epoch. E.g. assuming the list ranges from 8 to 16 peers

Fn submit(payments: Vec<MicroPayment>, ledger: &mut Ledger) // a server node submits accumulated micropayments to layer 1 ledger. The length of payments is the number of clients it serves during one block time.

Fn collect_fee(account: &Account, payments: Vec<MicroPayment>, ratio: f32) // the smart contract in layer 1 ledger will burn X% (X%=ratio) part of the payments that a node collected, and deposit the rest 1-X% of micropayments into the node's account.

Fn reward(ledger: &Ledger, n: &mut Node, payments:Vec<MicroPayment>, amt: u32) // layer 1 distributes rewards to individual nodes after a new block is mined. Here the reward depends on the micropayments the node submits to the ledger.

PoC security

Assume a server node collects payments=[p1,p2,...,pm] from m clients during one block time and it receives reward r after the block is finished. The net reward is given by: net = r + (1-x)*(p1+p2+...+pm)

Now we analyze the security of PoC for the most common sybil attack. Assume the malicious devices a party can control is X (e.g. X = 10%) of the total number of devices. During one epoch, the probability a malicious node has one malicious neighbor is close to X. The probability that it has two or more malicious neighbors is much lower. Assume this node cannot provide service by refusing all other peers but just collect fees from its malicious neighbor. In this case, the net reward is given by:
Net = r + (1-x)*p1

Simple design 1: Notice that the reward r is a function of [p1,p2,...,pm]. We define r = 0 if m=1. In this case, the reward of malicious node is negative (i.e. -x*p1) while the reward of honest node is positive (i.e. (1-x)*p1).

Thus, we only need to design the reward function that rewards more when m is larger.The reward function should be designed in such a way that incentivizes the node to serve more clients and the burned ratio will be compensated when it serves multiple nodes.

In practice, we will not store the micropayments history into blockchain. Instead, the reward will be a function of credit score and credit score should take the [p1,p2,...,pm] into consideration to reflect the above design principles.

Simple design 2: We can also remove the burning of micropayment. In this case, we will not increase the credit score of the server node if it only serves one client during one block. In this case, we rely on the fact, the probability to match two or more malicious nodes is rare.

In the above analysis, we assume the X is the ratio of malicious devices over all devices. In order to control this number, we require all the devices to deposit some tokens before joining the network. Thus, if the malicious party wants to create a lot of virtual devices,, it has to deposit a large amount of tokens which it would rather follow the protocol and play honest to earn the ming reward.