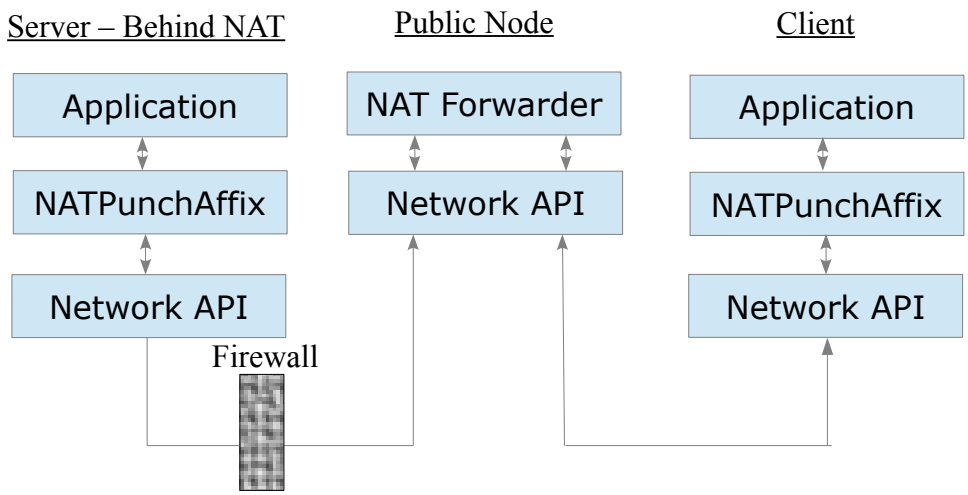
Use Cases for AFFIX

If you have already learned about the AFFIX framework and are familiar with the infrastructure, you may be wondering what are some use cases for this framework. You've had some time to play around with the awesome project and are excited about using it, but are not sure when or why you would need to use it. Don't worry! Below we provide you with some scenarios where using the AFFIX framework will make your task much easier! We start off with some simple situations and slowly build on them to show you some complex scenarios. In some cases we guide you on how to use simple AFFIX components to help you resolve simple problems, in other cases.

Node behind a NAT :

Home routers and firewalls have become very common now and have brought on some interesting challenges. One of the challenges that application developers face is how to circumvent a NAT such that the application works smoothly. The AFFIX framework provides an easy solution and allows the developer to provide NAT traversal for their application without having to make major modification to their code. We have provided a simple AFFIX component named NatPunchAffix which is an implementation of NAT traversal that is very similar to TURN. The NatPunchAffix allows the application behind a NAT to connect to a third node outside of the node allowing any client to communicate to the application behind the NAT through the third node.

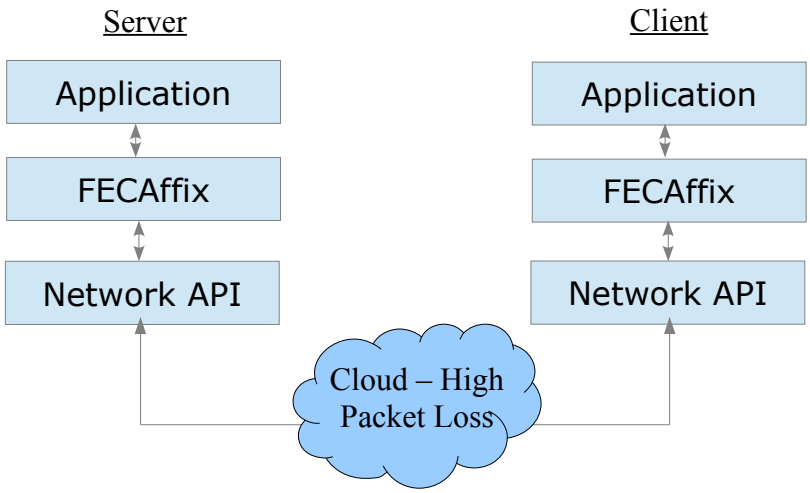


Details: When the application developer decides to integrate NAT traversal with their application, they can use the AFFIX framework along with the NatPunchAffix. Furthermore the developer can add in a NatDeciderAffix onto the AFFIX stack that determines whether or not the node is behind a NAT and if the NatPunchAffix needs to be used. Once the application behind a NAT opens up a listening socket and has determined that the NatPunchAffix needs to be used, the AFFIX initially looks up through a dht service all the available nodes that are running the NAT forwarder service. The NatPunchAffix will then randomly choose one of the available forwarders and register itself with it. The application will also publicly announce that it is using the NatPunchAffix and the address of the NAT forwarder service that it is using. If a client wants to connect to the server, it will initially contact the advertise service in

order to find out if the server is using the AFFIX framework, and if it is, which AFFIX components are being used. Once the client has identified that the server is using the NatPunchAffix and which NAT forwarder service is being used, the client will connect with the NAT forwarder service on the public node. Thus the NAT forwarder service will act as a relay between the client and the server and forward any messages received from one side to another.

High packet loss:

Packet loss could be a major issue for an application sometimes. Any portable device with a poor wifi connection could often suffer high data loss. If you are a software developer developing a new cutting edge media streaming application that supports multiple platforms, packet loss would be a major concern. There are various methods of packet loss mitigation that developers can implement, however it may require major change to the code base. The AFFIX framework developers to implement a simple AFFIX component that can help reduce packet loss. A simple component has been developed by the developers of the AFFIX framework named FECAffix, which uses the method of forward error correction (FEC) to help reduce the packet loss.

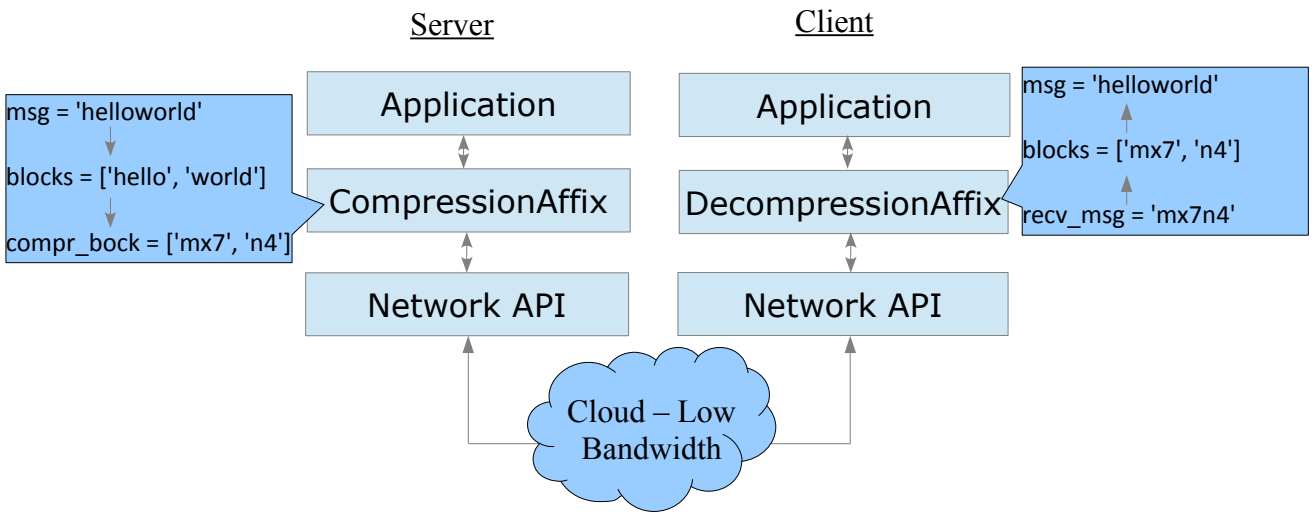


Details: The AFFIX framework can be used to easily reduce packet loss by using the FEC AFFIX component that we provide. The current implementation of the FEC AFFIX transmits an extra packet for every two packet that is sent out by default (this can be changed). The third packet has enough information about the first two packet, such that if one of the original two packet is lost, the receiving end is able to use the third packet to recover the lost packet thus reducing loss rate. Application developers are not required to use our FEC AFFIX that we provide and are more then welcome to build a more complex AFFIX component, if they wish to. The FEC AFFIX was build to demonstrate how easy it is to develop a component using the AFFIX framework to mitigate data loss over the network.

Node with low bandwidth:

In the modern day we often tend to take bandwidth for granted. Only a decade ago it was normal to see people running on a 56K modem. Even though we have come a long way from then and much improvement has been made, many ISPs in developing countries only offer limited bandwidth to residential connections. Furthermore with the explosion of so many different mobile devices, it is very

common to see people running their devices on wifi or 3G/4G which may have limited bandwidth. The network may be a bottleneck for many applications, especially if the device has decent cpu power. If networking is the bottleneck of the application then a simple solution would be to compress the data before transmitting it. For UDP packets this is a simple matter, however for TCP data flow compressing the data is a bit more difficult as it is a stream of data. The data stream needs to be split up into certain block sizes before they are encrypted and transmitted. With the AFFIX framework, the developers don't have to worry about all the small details when developing the application. Developers are able to easily separate out the network functionalities from the main application and use the AFFIX framework to handle networking challenges. We provide a simple Compression AFFIX that compresses the data before transmitting. Since the AFFIX framework lies underneath the application layer, the application itself is unaware of the data compression that occurs underneath it and does not have to worry about it.

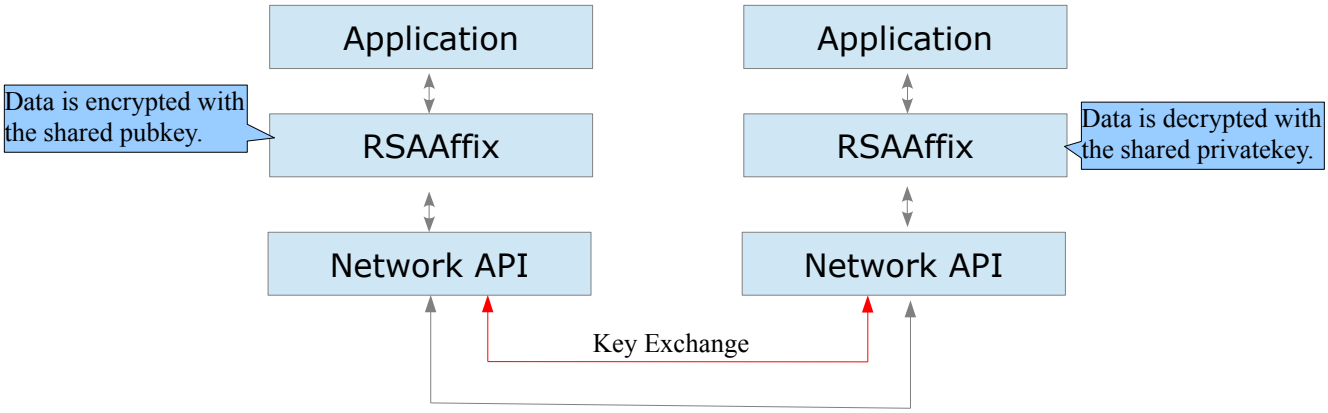


Details: The compression AFFIX takes messages and breaks them down into smaller blocks (1024 bytes by default). Each individual block is then compressed by the sender and added to a queue to be transmitted over. The sender also attaches the length of the compressed block so the receiver knows how much data to read in before uncompressing. Once the receiver receives the compressed data, it uses the information in the data to figure out the size of each compressed block and uncompresses each block. The uncompressed blocks of data is then combined before returned to the application layer.

Securing data transmission:

Security may be a major concern for application developers. Among many other security practices, encrypting data before transmission is often a good idea. This is especially useful if the application is transmitting sensitive data such as username/password/credentials and the developer wants to protect the data from eavesdroppers or man in the middle attacks. With the AFFIX framework, an application developer can easily add in multiple layers of security, each of which performs a different task. We provide two very simple AFFIX components that may be used to encrypt the data. The first component is AsciiShiftingAffix, which uses a simple Caesar cipher to shift each character of the data. This is a very simple component that shows application developer how easily developers can develop their own

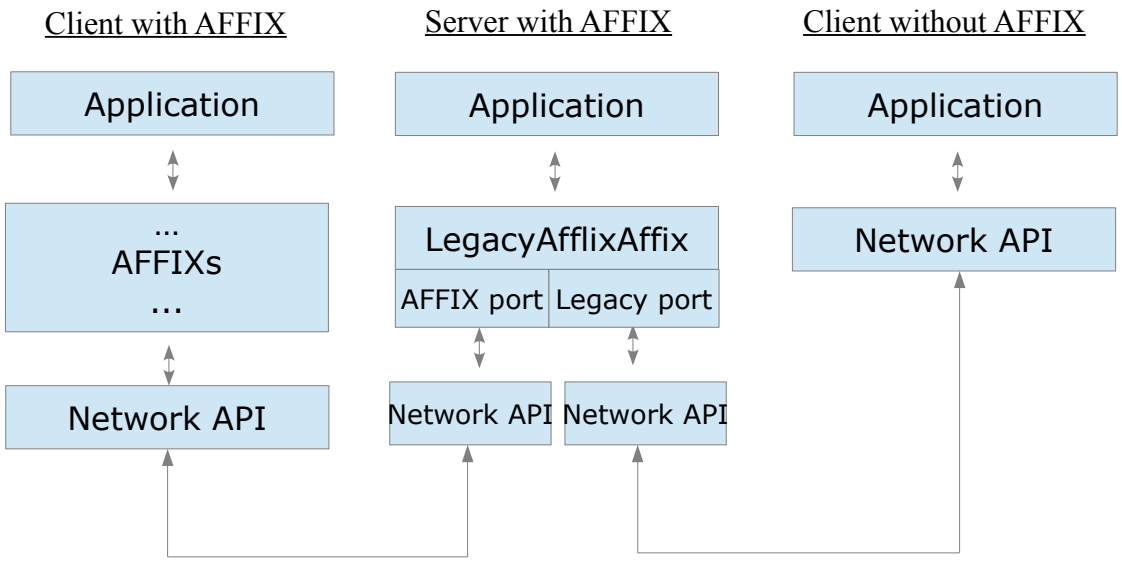
AFFIX component and plug it into the AFFIX framework. The second component we developed is the RSAAffix which uses the rsa library to generate new public/private keys and uses them to encrypt/decrypt data that are being transmitted.



Details: The RSAAffix generates a new set of public/private key pairs for each new connection that is made between the server and the client. The client originally generates a set of temporary key and sends the public key to the server. The server then generates a new set of keys on it's side. The server then encrypts the new generated public key with the temporary public key that the client had sent. The server then sends back the encrypted new public key to the client. The client can then use it's own temporary private key to decrypt the public key that the server had sent it. From this point onward, both the server and the client uses the new set keys to encrypt all communication between them.

Communicating with legacy clients:

A common question that an application developer might ask is how to coordinate between two nodes if one side is not using the AFFIX framework. As seen in some of the earlier examples, many of our AFFIXs need to be balanced otherwise the communication would break down. This could be an issue if the client is not using the AFFIX framework. This could be the case if an application has recently adopted the AFFIX framework and the has integrated the framework into the servers. For such scenarios we have provided an AFFIX component called the LegacyAffix, which allows servers to handle both clients that are using the AFFIX framework as well as legacy clients that have yet not adopted the framework. The LegacyAffix usually lies underneath the CoordinationAffix in the AFFIX stack.



Details: The legacy AFFIX allows both legacy clients as well as clients using the AFFIX framework to connect to a server that is using the AFFIX framework. The LegacyAffix opens up two different listening sockets, one for each type of clients. Both the sockets listen constantly in parallel for incoming connection. Usually the socket that is listening for AFFIX connections uses a higher port number then the socket listening for legacy connection. The Coordination AFFIX is responsible for advertising the ports that are being used by the AFFIX socket such that any clients that are using the AFFIX framework are able to connect to the proper listening socket on the server. The LegacyAffix is considered to be a branching AFFIX as it creates two branches of stacks underneath it.

Controlling data transfer:

Combination of NAT traversal, compression and encryption:

Using multiple interface along with bandwidth control: