Firewall and Service Tickets (FAST)

A means to request, provide, and monetize network services for the mutual of benefit application providers, network operators, and users

August 13, 2018

Tom Herbert
CTO, Quantonium
Santa Clara, CA
tom@quantonium.net

Executive summary

Emerging network access architectures and technologies include a wide range and rich set of capabilities to provide network services. While low level technology defines these capabilities, realizing the benefits all the way up to the end user is proving difficult. There are a number of roadblocks to overcome: 1) fine grained service differentiation is a hard problem 2) network state doesn't scale 3) monetization of services needs to be considered.

Firewall and Service Tickets (FAST) is a solution to help (everyone) realize the benefits of network services. The basic idea is that applications signal the network for the services they want applied to packets. This signal is encoded in a *ticket* that indicates the services that the network has agreed to apply to an application's packets. Applications request tickets from a *ticket agent* in the network. The ticket request is an abstract description of an application's traffic without divulging any specifics about the content. Issued tickets are attached to packets, processed by the network, and are reflected by peers so that they can provide similar functionality to stateful firewalls but in a stateless fashion. Since tickets are attached to every packet, a *pay per use* charging model is achievable with proper accounting.

Example: network services in 5G

5G, the next generation mobile standard currently being developed by the 3rd Generation Public Partnership (3GPP), provides a good example of the opportunities in reaping the benefits of network services. A key feature introduced by 5G is the notion of a network slice. A network slice is a virtualized network that is defined to run over a physical network but can have its own operational characteristics. Network slices, in combination with Network Function Virtualization (NFV), provide the foundation for a rich and diverse set of network services that could provide low latency, high throughput, optimized mobile routing, etc. Operators are eager to make use of new technologies for network services with the intent of improving user experience and getting return on investment for 5G.

In order to fully utilize network services, traffic from different applications needs to be differentiated per the services they need. To differentiate traffic, the network needs to deduce the characteristics of an application based on the packets it sees and then apply the appropriate services. This process is called *service mapping*-- packets sent by an application are mapped to the network services that the application wants applied. Service mapping happens today, however the techniques used are *ad hoc* and imprecise. Applications don't generally give obvious or reliable hints to help the network do service mapping. Application payloads, such as HTTP, are increasingly being encrypted to hide content. Differentiated services set in IP headers are coarse and can too easily be cheated.

Mapping based on destination address provides benefit to only select content application providers that have special arrangements with operators.

The desired solution is that applications explicitly request the services to be applied to their traffic. In a 5G network, this would entail that applications running in UEs (User Equipment) indicate desired services to be provided by the RAN (Radio Access Network) and core network. For instance, a video chat application may request bounded latency that is implemented by the network as a network slice; so packets sent by the application should be mapped to that network slice.

Goals and requirements

The solution to service differentiation is for applications to signal the network for the services they want. This has to be done in a way that is generic, unbiased, and allows fair monetization.

A solution should meet the following requirements:

- Must be stateless within the network. Intermediate nodes must not be required to create and maintain state for transport layer connections.
- It must work in multi-homed and multi-path environments.
- The solution doesn't leak internal information about network internals to untrusted parties in the outside world.
- A solution must work with any transport protocol as well as in the presence of any IP protocol feature (e.g. in the presence of extension headers).
- The solution does not require deep packet inspection.
- Solution should minimize changes to an application.
- The solution must deter misuse that might result in illegitimate use of network services or denial of service attack.
- A network needs to apply services to both directions of a communication. How services are applied in the return path of a flow must be considered.
- In circumstances where the mechanisms of a solution are problematic for use on the Internet, there must be a fallback that allows functional communication but potentially in a degraded mode of service.

Current approaches

Stateful firewalls and *transport layer proxies* are the predominantly deployed techniques to control access to a network and map packets to services on a per flow basis. While they provide some benefits of security, they also break the end-to-end model and have otherwise restricted the Internet in several ways:

- They require parsing over transport layer headers in the forwarding path.
- They are limited to work only with a handful of protocols and protocol features.
- They break the ability to use multi-homing and multi-path.
- They break end to end security. NAT, for instance, breaks the TCP authentication option.
- They create single points of failure and have become network bottlenecks.

Both IPv4 and IPv6 have specified a field in the IP header for signaling quality of service (QoS) to the network. In IPv4 this was referred to as the Type of Service (TOS), and in IPv6 the it is called Traffic Class. These fields have be overloaded in time to hold *differentiated services* (*diff-serv*) values. Differentiated services provides an IP layer means to classify and manage traffic, however it is lacking in richness of expression and a ubiquitous interface that allows applications to request service with any granularity. Diff-serv is useful in closed networks where all parties can be trusted, but in the general Internet diff-serv lacks security and uniformity in how its being set to be useful.

Some network devices perform *Deep Packet Inspection* (*DPI*) into the application data to classify packets to determine whether to admit packets or what services to apply. For instance, HTTP is commonly parsed to determine URL, content type, and other application related information. DPI is only effective with the application layer protocols that a device is programmed to parse. More importantly, application level DPI is being effectively obsoleted in the network due the pervasive use of Transport Layer Security (TLS).

Alternative proposals

SPUD (*Session Protocol Underneath Datagrams*) and its successor *PLUS* (*Path Layer UDP Substrate*) proposed a UDP based protocol to allow applications to signal a rich set of characteristics and service requirements to the network.

SPUD had a number of drawbacks:

- It was based on a specific protocol used over UDP, so SPUD was incompatible with TCP which is the predominant transport protocol on the Internet.
- SPUD required that intermediate nodes parse and process UDP payloads based on matching port numbers to applications. Port numbers are not reserved numbers with global meaning, so processing a UDP payload based on port numbers can lead to misinterpretation or even silent data corruption.
- SPUD included stateful flow tracking in the network. Not all communications have stateful semantics, stateful devices break multi-homing, and maintaining state in the network is hard to scale.

- The meta-data information in SPUD would need global definition. This would leak application specific information to untrusted parties, and establishing a standard on what data is universally relevant would be difficult.

Segment routing is a recently defined technique that proposes using an IPv6 routing header to source route packets through a network. This allows "network programming" where packets can visit various nodes towards the destination, each of which may apply some Network Function Virtualization (NFV). A segment routing header is composed of a list a "segment identifiers" (SID) in IPv6 address format. SIDs can either be addresses of intermediate destinations, or instructions interpreted by intermediate nodes.

Segment routing for service mapping has some drawbacks:

- Segment routing is intended to be confined to a segment routing domain. The protocol is not designed for use over the Internet.
- The protocol is verbose. Each SID is sixteen bytes which adds up to considerable overhead when even a moderate number of SIDs are present in a packet.
- Segment routing is explicit so there's little flexibility at intermediate nodes to reroute packets around bottlenecks are failed nodes.
- The segment list is visible to both sender, receivers, and all intermediate nodes so that sensitive internal information may easily be leaked.
- Segment routing only conveys information about the forward path of packet, it says nothing about how apacket is to be routed in the return path.

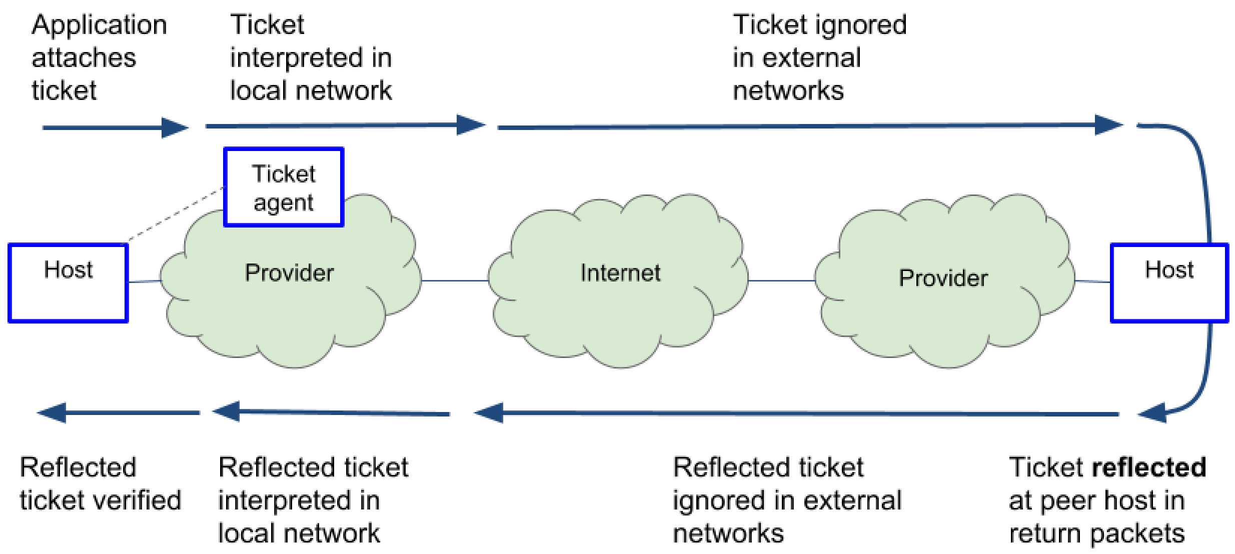
Firewall and Service Tickets

Firewall and Service Tickets (*FAST*) is a facility to allow an application to signal to the network requests for admission and services for a flow. A ticket is data attached to a packet by the source that is inspected and processed by intermediate nodes in a network. Tickets express a grant or right packets to traverse a network or have services applied.

An application requests tickets for admission or services from a ticket agent in their local network. The agent issues tickets to the application which in turn attaches these to its packets. In the forwarding path, intermediate network nodes interpret tickets and apply requested services.

Tickets are validated for authenticity by the network and contain an expiration time so that they cannot be easily forged. Tickets do not have a global interpretation, they are only interpreted within the network that issues them. To apply services to inbound packets for a communication, remote peers reflect received tickets in packets they send in the reverse path. Tickets are stateless within the network, non-transferable, and revocable.

The figure below illustrates the use of tickets from an application on a client host to a server in the Internet.


Application attaches ticket
Ticket interpreted in local network
Ticket ignored in external networks
Ticket agent
Host
Provider
Internet
Provider
Host
Reflected ticket verified
Reflected ticket interpreted in local network
Reflected ticket ignored in external networks
Ticket **reflected** at peer host in return packets

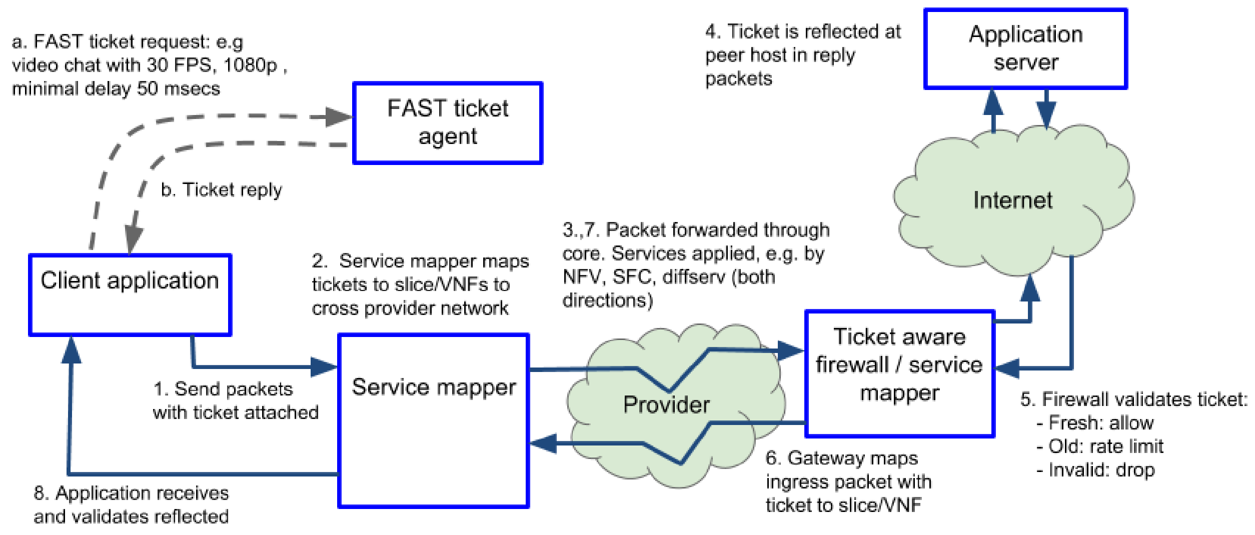
Within a provider network, services may be provided on behalf of users of the network. In the figure above, the provider on the left may provide services and service agreements for users in its network; and likewise the provider on the right can provide services to users in its network. Transit networks service all users and don't typically provide user specific services or service differentiation. Services provided by different provider networks may be very different and dependent on the implementation of the network as well as the policies of the provider.

The use of tickets may be bilateral for a flow so that each peer requests service from its local network. Therefore packets may contain two types of tickets: one that is set by the sending host to signal its local provider network, and the other is the reflected ticket that is a signal to the provider network of the peer.

Tickets are scoped values, they only have meaning in the network in which they were issued. The format, meaning, and interpretation of tickets is network specific. By mutual agreement, two networks may share the policy and interpretations of tickets. For instance, there could be an agreement between two provider networks to interpret each others tickets or to use a common format.

Video chat example

The diagram below illustrates an example of FAST being use to provide network services for a video chat application. The are services might be low latency and high throughput.


a. FAST ticket request: e.g video chat with 30 FPS, 1080p , minimal delay 50 msecs
FAST ticket agent
b. Ticket reply
Client application
1. Send packets with ticket attached
2. Service mapper maps tickets to slice/VNFs to cross provider network
Service mapper
8. Application receives and validates reflected
3.,7. Packet forwarded through core. Services applied, e.g. by NFV, SFC, diffserv (both directions)
Provider
4. Ticket is reflected at peer host in reply packets
Application server
Internet
Ticket aware firewall / service mapper
5. Firewall validates ticket:
- Fresh: allow
- Old: rate limit
- Invalid: drop
6. Gateway maps ingress packet with ticket to slice/VNF

In this example, the client application requests a ticket from the local ticket agent. The request indicates characteristics of the communication such as frame rate, latency, jitter, etc. Note that the request does not indicate what the content of communication is. The FAST ticket agent in the network receives the request and determines if the it can be met. This may include applying policy, capability determination of the path, or charging limits. If the request can be met, then a ticket is issued. Tickets normally have an expiration time, so the request might re-evaluated again in the future. In this way, the tickets could be used as a soft method of network resource reservation.

When a ticket has been issued to an application, it is attached to packets sent by the application. The first hop node in the path of a packet processes the ticket and and applies the appropriate services. The network services may be instantiated by diff-serv marking, forwarding on a network slice, routing to a VNF, encapsulating in MPLS, etc.

Packets traverse the local network with the appropriate services being applied. When they leave the local network, the tickets still are present, however external nodes don't process the them; tickets are opaque outside of the network they originated in.

Eventually packets arrive at the peer destination server. The peer host observes an attached ticket in a received packet and saves the attached ticket in the context of the flow for the communication. The peer application doesn't interpret the ticket, it merely saves it.

Subsequently, when the the peer sends back towards the client, the saved tickets are attached to packets as reflected tickets.

Similar to tickets in the forward path, reflected tickets are ignored in external networks on the reverse path until a packet reaches the origin network of the ticket. An ingress node in the client's network interprets reflected tickets and applies appropriate services for traversing the local network. Packets are forwarded in the client network with appropriate services applied. In this manner, video from the remote user to the client gets appropriate service in the client's network without network state being maintained.

Eventually, a packet with a reflected ticket is received by the originating node. The reflected ticket is validated by comparing the received ticket with that being sent for the flow. If the ticket is determined valid then the packet is accepted.

FAST details

FAST tickets are encoded in IPv6 Hop-by-Hop options. The format of a ticket is defined by the network in which the ticket is issued. A ticket should be obfuscated or encrypted for privacy so that only the local network can interpret it. It should be resistant to spoofing so that an attacker cannot illegitimately get service by applying a ticket seen on other flows.

Ticket encoding in IPv6 Hop-by-Hop option looks something like this:

Option type	**Data length**	**Type**	**Reserved**
Ticket			

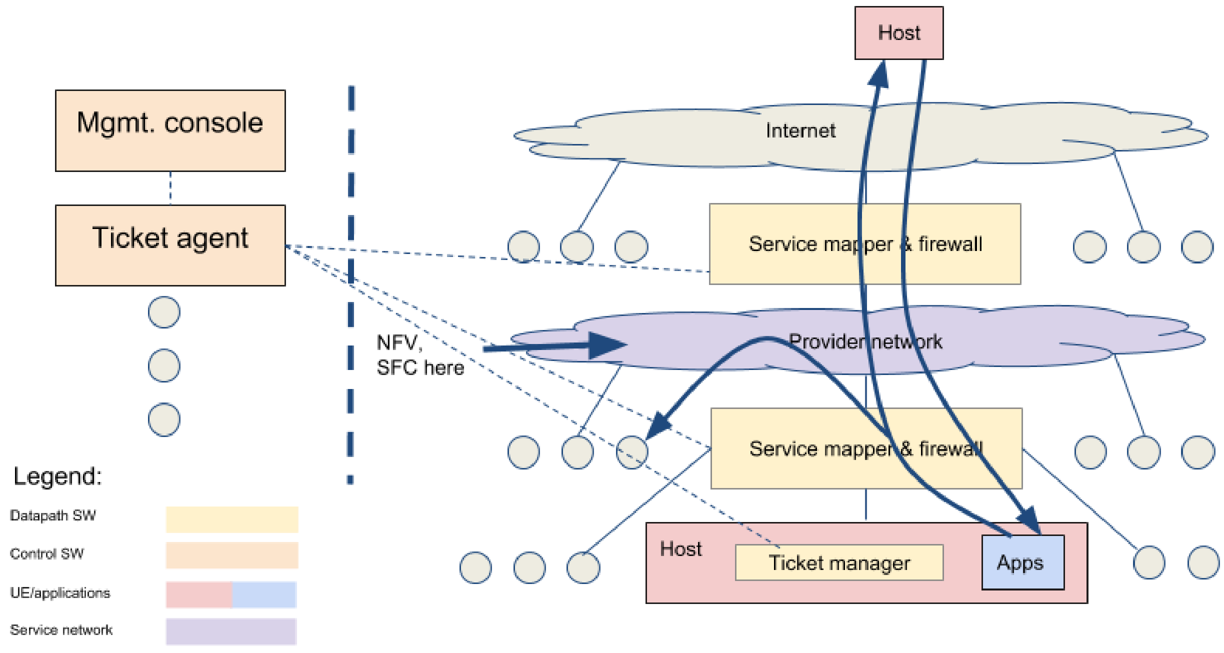
It is expected that tickets are encrypted and each ticket has an expiration time. For instance, a ticket may be created by encrypting the ticket data with an expiration time and using the source address, destination address, and a shared key as the key for encryption. The operative part of ticket that describes service may have different types of data. For instance, a set of flags could be used, a list of service values, or a profile index into a table that describes a set of services.

A simple unencrypted ticket containing a typed service protocol index might look like:

Expiration	
Verification data	
Type	**Profile index**

FAST Implementation

The diagram below illustrates the network components relevant to FAST.



The relevant components for FAST are:

- Applications
- The client OS (kernel and userspace OS libraries)
- Service mappers
- Ticket agent and management
- Internet servers

Existing client *applications* can be modified to request tickets and set them in packets. It is also possible for the OS to set FAST tickets on behalf of an application that can't be changed recompiled. The kernel may need some small changes or configuration to enable an application to specify the FAST Hop-by-Hop option for its packets. In BSD sockets this can be done by a *setsockopt* system call or in ancillary data of the *sendmsg* system call.

An application that wishes to use network services first requests tickets from a ticket agent. The request could be in the form of an XML structure with a canonical elements. The request could be sent via a web service using RESTful APIs. Internally in the host, the ticket agent might be accessed through a library that interfaces to a ticket daemon that in turn arbitrates requests between applications and a ticket agent in the network.

Service mappers need to parse and process tickets in the fast datapath. This entails an implementation that does efficient IPv6 header processing. Tickets need to be parsed, validated or decrypted, looked up, and interpreted quickly.

To perform ticket reflection, servers must be updated. In the case of a connected socket (TCP, SCTP, or a connected UDP socket) this is a relatively minor change to the kernel networking stack which would be transparent to applications. For unconnected UDP, an application could use ancillary data in recvmsg and sendmsg to receive and reflect tickets.

Monetization and policy

Tickets facilitate fine grained policies and "per use charging" of services. There are three points at which policy and charging can be applied: 1) at ticket request time 2) when a client sends a packet with a ticket 3) when a network ingress node receives a packet with a reflected ticket

At ticket request time, policy can be applied to determine if the network can or should provide the services being requested. Ticket requests are authenticated and the requestor's identity is known. A database can be maintained that holds the per user policies, resource limits, and current resource utilization as input to a policy decision for issuing a ticket.

Each time a ticket is seen in the network it can be accounted for. The two cases, when a ticket is sent from a client or a ticket is reflected by a server, should be accounted separately since the tickets sent directly from the client are a bit more trustworthy. A rogue peer, for instance, could attempt a narrow DDOS attack by flooding the flow with fake packets using the same ticket.

Per use accounting is done at the service mappers. Each occurrence bumps a counter. Aggregated counts are periodically sent to a centralize accounting system that correlates the use of ticket across the service mappers. Based on the resultant data, users can be charged precisely for the services they actually used.

Conclusion

Firewall and service tickets is part of a larger effort to bring the Internet into the 21st century. It promotes the end-to-end principle, net neutrality, and clean protocol layering. It also leverages forward looking features of the IPv6 protocol. The ultimate goal is to spur innovation in network services for all users with fair monetization. For more information about the protocol , see https://datatracker.ietf.org/doc/draft-herbert-fast/.