MobilityFirst Protocol Stack

The MobilityFirst protocol stack captures the following core architectural principles:

- Applications should speak to high-level, abstract identifiers that are independent of network location
- Applications should be given the ability to state *routing preferences*; that is, how routers should treat the message when network-level decisions must be made. These preferences help capture whether the traffic is real-time or delay-tolerant.
- A thin, possibly reliable, transport protocol is favored over today's connection-oriented protocols.
- Large blocks of the message, called *chunks*, are reliably transmitted in a hop-by-hop fashion. A chunk is an autonomous unit of information from a routing perspective, and is labeled by a *MobilityFirst GUID-based header*.

MobilityFirst maintains traditional layering, with the exception of an optional end-to-end security layer that is given the role of performing GUID-based security operations.

The following table illustrates the labels and data units understood by each of the layers.

Layers of the MobilityFirst stack:

Layer	Data Units Understood	Labels Understood
MF Application	Messages	GUIDs
MF E2E Security	Messages	GUIDs
MF Transport	Messages & Chunks	GUIDs
MF Routing	Chunks	GUIDs, Net Addresses & MACs
MF Link	Chunks & Frames	MACs
MF Physical	Frames	MACs

The following three layers are present on all MobilityFirst clients, and may or may not be present on MobilityFirst routers: (1) application, (2) E2E security, and (3) transport.

The *application layer* understands only messages destined to GUIDs. Therefore, the traditional socket interface can be replaced by one that includes, at minimum, a sendTo operation where the destination is a GUID. Human-readable names, such as "Jon's laptop" or "ESPN's highlight video" will be mapped to GUIDs via application-layer services for context, content, and devices.

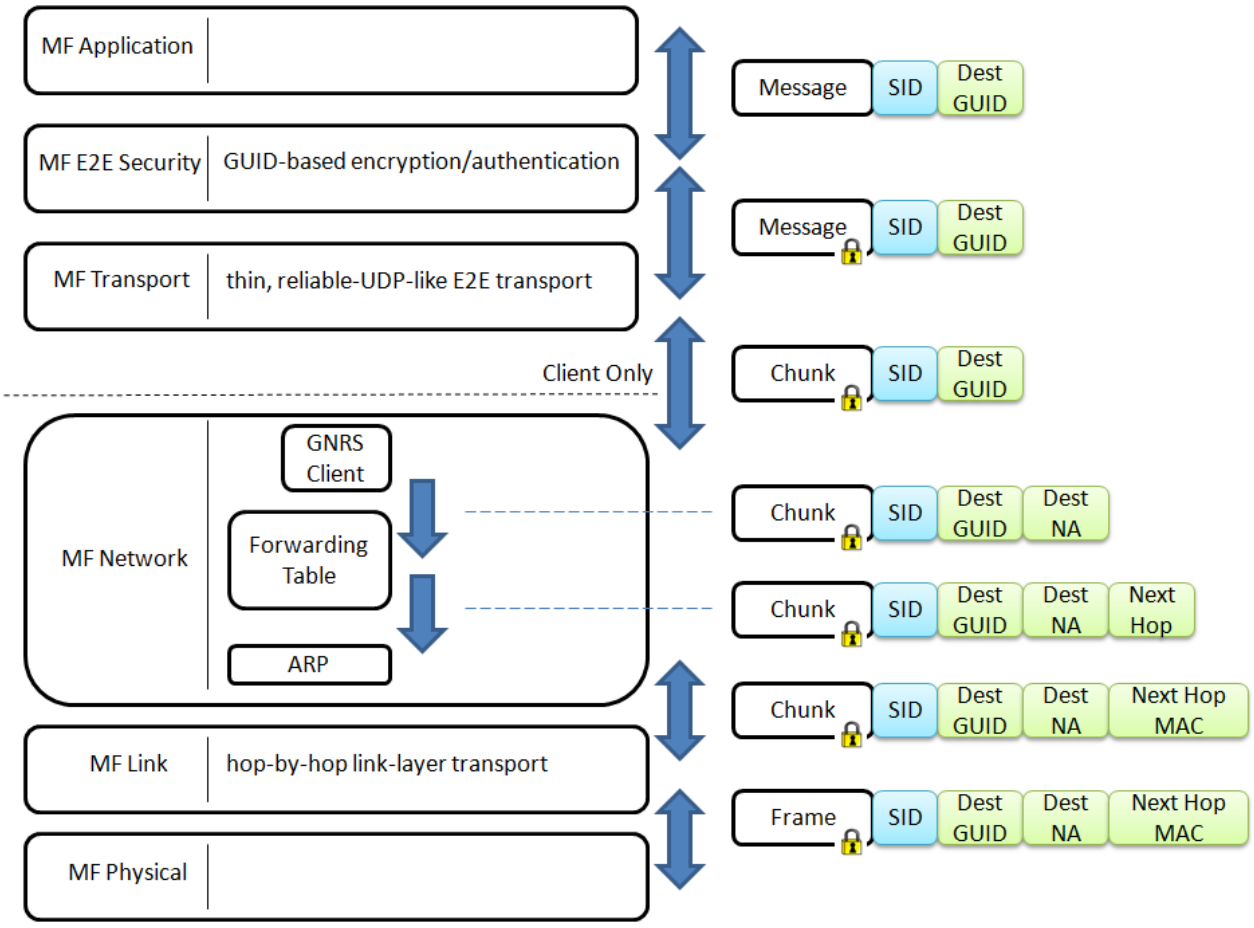
The *E2E security layer* is responsible for performing all GUID-based security services. This layer stores the private keys to all GUIDs owned by the device, and can sign messages on behalf of

applications. Furthermore, it can encrypt messages for a foreign GUID using the destination GUID provided by the application. Finally, it can verify signed messages destined for a local application, as well as decrypt messages send to one of its GUIDs. This layer may also be responsible for storing and verifying certificates via a collected set of root certificate authorities. In this case, it takes some of the functionality from today's browser-based, TLS security.

The *transport layer* provides a socket interface for applications and the security layer to access the lower-level network layers. This layer is responsible for taking a message and breaking it into large *chunks*, reliably transmitting the chunks (if the service ID desires this), and re-compiling the chunks into a message at the destination.

The following diagram illustrates data traversing the MobilityFirst protocol stack:

Data traversing the MobilityFirst stack:



The lower three layers are present on all MobilityFirst routers: (1) network, (2) link, and (3) physical.

The *network layer* is responsible for taking a chunk and adding a *next hop* field to it that the link layer can understand. Under normal operation, a *GNRS client process* will run at this layer, taking a GUID and making a GNRS query on that GUID, in hopes of receiving a list of NAs corresponding to that GUID. It may also receive a list of *local addresses (LAs)* that provide the local address of the GUID which is understandable by the intra-domain routing protocol. In the event that either the GNRS is inaccessible, or the destination GUID is inaccessible, or the local GNRS returns a LA that is not part of the intra-domain forwarding table, the routing protocol can attempt to directly route on the GUID itself. This is usually the case in DTN environments. Either via the LA forwarding table or the GUID forwarding table, the router will append a *next hop MAC address* to the chunk and pass it to the link layer. Note that the routing layer has direct access to a large storage buffer, and can make the decision to temporarily store chunks in this buffer, according to the rules of the intra-domain routing protocol. It should, however, respect the application preferences set in the SID regarding storing chunks.

The *link layer* is responsible for reliably transferring a chunk by breaking it into smaller frames and transferring them to the next hop. The next hop's link layer must completely and reliably obtain the chunk before it informs the routing layer that it has received the chunk. If the chunk, for whatever reason, cannot be transmitted, then the sending side link layer must give the chunk back to the routing layer, who can then either re-route or store the chunk if need be.

Finally, the *MF physical layer* is responsible for delivering a single frame to the next hop.

The following table provides an idea of the interfaces between the layers:

Interface for layers:

Layer	Going down the stack - Called from above	Going up the stack - Called from below
MF Application	N/A	giveMessageToApp(Message, from GUID)
MF E2E Security	sendSecMessage(Message, to GUID, sec) getSecContent(content GUID, sec)	giveSecMessageToApp(Message, from GUID)
MF Transport	sendMessage(Message, toGUID) getContent(content GUID)	giveChunkToTransport(Chunk, from GUID)
MF Routing	sendChunk(Chunk, to GUID)	giveChunkToRouting(Chunk, from MAC)
MF Link	sendChunk(Chunk, next hop MAC)	giveFrameToLink(Frame, from MAC)
MF Physical	sendFrame(Frame, next hop MAC)	N/A

Note that the *service ID* (SID) will specify whether or not the application desires any part of the message to be stored, as well as whether it is real-time traffic or not. This directive impacts the transport, routing, and link layers. The transport layer will not reliably transmit real-time traffic, and will create much smaller chunk sizes for it. The routing layer will follow the storage directives when making routing decisions. The link layer may attempt to reliably transmit real-time data, but will give up more quickly.