TCF UDP Discovery

Designed by Eugene Tarassov (eugene.tarassov@windriver.com)
Documented by John Cortell (john.cortell@freescale.com)

Created on: December 5, 2009
Last modified on: May 24, 2010

Overview

TCF Discovery for agents connected to a network is done via UDP. By default, a static, well-known (to TCF agents) port is used: 1534. However, we will see that agents on a machine can participate in discovery even if that port is unavailable; all that is required is that at least *one* agent on the network is listening on the well-known port.

Fundamentally, TCF discovery is based on agents advertising their peers by sending UDP packets to other agents, both proactively and reactively. This concept is simple enough when thinking in terms of one agent per machine. However, there can be multiple agents per machine and that significantly complicates the discovery mechanism as there can only be one process listening on a given port for a given protocol (UDP or TCP). That means only one agent per machine (NIC, actually) can service discovery requests on the well known port. How do other agents running on that machine get in the game? The TCF discovery mechanism addresses this using the notion of master/slave. Basically, the first agent to acquire the well-known port (1354) becomes the master. All other agents on that machine become slaves.

The master/slave relationship in TCF discovery is not what one might intuitively think, though. Masters and slaves are not all that different from each other. A slave is not exclusively reliant on the master on its machine. It can rely on any master on the network (subnet, actually; more on this later). Moreover, a slave's reliance on masters is limited. Ultimately a slave agent ends up establishing a *direct* relationship with every other slave on the network, and of course has access to the masters through UDP broadcasting (to port 1354). The role of the master is simply to make it possible for a new-comer slave to initially meet other slaves. Ultimately, every slave meets every other slave, and they all directly deal with each other thereafter.

This approach may seem a bit strange. A simpler and more intuitive approach would be to have the master on a machine be the representative for the slaves on that machine. A slave would just communicate locally with its master through UDP unicasting, and the masters would communicate among themselves through UDP broadcasting, proliferating information about themselves and their slaves. However, such an approach would make vendors too vulnerable because the master agent is simply the first agent that is able to grab the well-known TCF discovery port. There are no guarantees that the agent will turn out to be a well-behaved one. What if its discovery logic is buggy? What if it hangs? What if it supports only version X of the discovery protocol but other agents on that machine rely on version Y? Any of these scenarios could render other vendors' agents useless on a machine. Since the TCF discovery mechanism does not employ dedicated reliable masters but rather relies on every agent volunteering for the role, then the mechanism must be robust enough to overcome poorly-behaved volunteers. That is accomplished by limiting the role of the master and having all slaves know about, and deal with, all other slaves directly. Slaves do not need to keep track of masters explicitly; an agent can reach any and all masters by broadcasting to the well-known port (1354). The end result is that a single master agent can make discovery possible for the entire network.

Scope

TCF UDP Discovery is limited to IP subnet scope by design. What that means is that an agent will discover only the peers available on the subnet(s) its machine is on. If an agent is running on a machine with two NICs, each on a seperate subnet, then that agent will discover peers on both subnets. However, such an agent will not act as a bridge for discovery across the two subnets. That is, if machine A is on subnet X only, machine B is on subnets X and Y, and machine C is on subnet Y only, agents on B will know the available peers on subnets X and Y, but agents on A will not know about peers on Y and agents on C will not know about peers on X.

For simplicity, further discussion on TCF discovery will just refer to agents "on the network", but keep in mind that subnet scoping is implied.

Rules of Engagement

Fundamental to the discovery mechanism is the notion that agents must work together to keep each other in sync regarding available agents and peers. More specifically, agents must be *inquisitive, responsive* and *proactive* in sharing its knowledge of peers and agents. An agent is *inquisitive* by asking other agents for their tables. It is *responsive* by providing their own tables when asked. Finally, it is *proactive* by volunteering that information even when not asked. This last element is necessary in order to quickly proliferate new information, without requiring constant polling by agents.

Of course, all this activity back and forth between many agents could easily get out of hand without having well designed *rules of engagement*. A reasonable balance of all three activities must occur. Too much interaction will lead to excessive network traffic and agent load. Too little will prolong stale/inaccurate information. The rules of engagement will be discussed shortly.

The final element of discovery is validation. As there are no notifications in TCF discovery for a peer or agent going away, agents must regularly validate their tables for accuracy and prune accordingly. This is also covered in the rules of engagement.

The following are the TCF UDP Discovey *Rules of Engagement.* Unless explicitly qualified, "agent" refers to both masters and slaves:

1. Agents should, on launch:
    - broadcast a request for peers to all master agents
    - broadcast descriptions of its peers to all master agents

2. Upon receiving a packet (any) from an agent, the receiving agent should ask the sender for its slave table, but should make such a request only occasionally for any given subnet. In other words, an agent should periodically ask another agent (any agent) for its slaves.

3. Once an agent receives a CONF_REQ_SLAVES from a slave, for duration of X seconds (retention period, see rule 5), the agent should thereafter provide that slave its slave table with every subsequent CONF_REQ_INFO it receives from that slave, and it should also proactively provide that slave info about all new discovered slaves during that time. Essentially, by sending CONF_REQ_SLAVES (rule #2), a slave creates a temporary coupling between it and that agent (usually a master),

where the agent (master) proactively updates the slave's slave table.

4. An agent should perpetually and periodically notify all known agents with the collection of peers it makes available. Master agents should additionally, but selectively, advertise peers from other agents. A master should advertise peers from local slaves to all known agents. A master should advertise peers from remote agents to local slaves via the loopback address. The main purpose of these additional responsibilities is for discovery to work in the presence of a firewall, as long as the firewall permits traffic through the well-known port (1354). A good interval for the notifications is 1/4 the *retention* period (see rule 5). At each interval, an agent should ensure that it has sent at least one UDP packet. If it has nothing to advertise, it should send a CONF_SLAVES_INFO with no data (just the header). Failure to regularly send something may result in the agent being forgotten by the other agents. A TCF agent can have nothing to contribute, but still be a consumer of discovery data. A slave agent will receive updated discovery data only if it sends something. A master agent must actively be involved in discovery; it needs to make its presence known to slaves, but slaves will see it only if it sends something.

5. Agents should have short term memory when it comes to its knowledge of other slaves and and peers. If it hasn't received a UDP packet from a slave in X amount of time, it should forget that slave. X is called the *retention period.* 60 seconds is a good, default value. Similarly, if it hasn't received a CONF_PEER_INFO packet for a peer in X amount of time, and it doesn't have an open channel to that peer (not actively using it), then it should dispose and forget that peer.

6. If a slave agent has not received a packet from the local master in a while, then it should attempt to become the master by opening the well-known port (1534). By definition, successfully opening that port makes that agent the master on that machine.

7. An agent is introduced to other slaves either by receiving another agent's slave table (CONF_SLAVES_INFO) or by receiving a packet directly from the slave. When an agent becomes aware of a new slave, it should immediately:
    - ask the new slave for its peers
    - tell the new slave about all known peers and slaves
    - advertise the new slave to all slaves that have sent CONF_REQ_SLAVES to this agent within the last X seconds (retention period)

UDP Packets

All TCF discovery UDP packets have an eight byte header:

- bytes 0, 1 and 2 are 'T', 'C', and 'F', respectively
- byte 3 is the discovery protocol version, currently 2, in ASCII
- byte 4 is the packet type
    - 1 = CONF_REQ_INFO
    - 2 = CONF_PEER_INFO
    - 3 = CONF_REQ_SLAVES
    - 4 = CONF_SLAVES_INFO
- bytes 5-7 are reserved for future use

All additional data is packet-type specific. All strings in that data are UTF-8 encoded.

CONF_REQ_INFO

This is a request for peers info. A recipient of this request should respond by sending back (to whomever sent the request) a CONF_PEER_INFO for every peer it wishes to advertise. This packet has no additional bytes besides the header.

CONF_PEER_INFO
This is a packet which advertises a peer. An agent will send this reactively in response to a CONF_REQ_INFO request or proactively as dictated by the rules of engagement. Following the eight-byte header, this packet contains data that describes one and only one peer. The description is simply the attributes of the peer (IPeer.getAttributes()), in the format *key=value*, with each attribute terminated by a zero byte.

CONF_REQ_SLAVES
This is a request for slaves info. A recipient of this request should respond by sending back (to whomever sent the request) one or more CONF_SLAVES_INFO packets to relay the recipient's slave table. This packet has no additional bytes besides the header.

CONF_SLAVES_INFO
This is a packet which advertises one or more slaves. An agent will send this reactively in response to a CONF_REQ_SLAVES request or proactively as dictated by the rules of engagement. Following the eight-byte header, this packet contains data that describes one or more agents. The format of an agent description is:

<timestamp>:<port>:<host><zero-byte>

Where

timestamp = indicates when a packet from this agent was last received, in Unix time (milliseconds since 1-1-1970)
port = the port the agent is listening on (for TCF discovery)
host = the IP address or host name the agent is listening on
zero-byte = single byte with value 0

An example:

1234567890:1940:suki.acme.com

This indicates knowledge of a TCF slave agent running at suki.acme.com, listening on port 1940, and that a packet was last received from it on February 13, 2009 at exactly 23:31:30 (UTC).

Multiple such descriptions can appear in a CONF_SLAVES_INFO packet. Agents should take care to not send excessively large UDP packets, though. The DSDP reference implementation keeps packets under 1500 bytes. Agents should split large tables across multiple UDP packets.

Timing
Frequency of discovery activity is just as important as the rules of engagement but are harder to define. The sweet spot for any particular proactive operation is subject to agent and network factors. We will here list values used by the Java TCF reference implementation hosted in the DSDP/TM project.

Retention Period

This is the maximum amount of time an external peer and slave should be remembered barring additional confirmation of existence. 60 seconds.

Temporary Master-Slave Coupling
By sending CONF_REQ_SLAVES (rule #2), a slave S creates a *temporary* coupling between it and the receiving agent A (usually a master), where A becomes responsible for proactively updating S's slave table. This temporary coupling lasts X seconds, where X = retention period (60 seconds)

Periodic Activity
Agents should proactively send peer info packets (see rules of engagement 4). It should do so periodically, and regardless of incoming activity (as opposed to being triggered by incoming packets). It should at the same time update its subnet list, as users on a machine can enable/disable/reconfigure NICs on the fly. Finally, it should perform housekeeping on its peer and slave table. 1/4 retention period (15 seconds).

Master Volunteering
An agent that has not received a packet from the local master agent for more than 1/2 retention period (30 seconds) should attempt to open the well-known port and become the master. It should do this check during its *periodic activity.*

Slave Table Requests
An agent should occasionally take an incoming packet as an opportunity to ask *any* other agent on the network for its slave table (see rule of engagement 2). The minimum wait time on making such a request depends on the nature of the agent who's packet was just received:

- agent is a slave (on this or another machine): 2/3 retention period (40 seconds)
- agent is the master of another machine: 1/2 retention period (30 seconds)
- agent is the master of this machine: 1/3 retention period (20 seconds)

Basically, slaves should be engaged less often than masters, and remote masters less than local masters.

Miscellaneous Implementation Considerations

1. Because a master agent not only responds to peer requests (on port 1354) but broadcasts them (to port 1354), it will receive its own requests. An agent should check and ignore any packets it has sent.

2. The DSDP/TM C implementation of the locator service uses slightly different symbolic names for the packet types:

Java	C
CONF_PEER_INFO	UDP_ACK_INFO
CONF_REQ_INFO	UDP_REQ_INFO
CONF_SLAVES_INFO	UDP_ACK_SLAVES
CONF_REQ_SLAVES	UDP_REQ_SLAVES