23

The Better Browser Bureau

> There is a finite amount of clue
> in the Universe...
> and the Universe is expanding.
>
> — Unknown
> (thanks to John Ladwig
> and Marcus Ranum)

Hold on to your hoopskirts everyone, we're not there yet. We have a few more things to learn about the Network Neighborhood.

23.1 Running an Election

Elections may be called whenever a Consumer is unable to find a Local Master Browser, or when a jealous rival (known as a Preferred Master Browser) shows up. An election can also be forced by sending a zero-filled `RequestElection` frame.

When a `RequestElection` frame is received by a Potential Browser (including Backup Browsers, the LMB, and the DMB), the Potential Browser switches into election mode. The browser stays in election mode until a winner declares itself by sending a `LocalMasterAnnouncement` frame.

While in election mode, the browser sends and receives `RequestElection` frames. If another browser's credentials are better, then the browser knows that it has lost the election and will politely shut up, not participating further in the current election.

23.1.1 *Voting*

There is a bit of timing involved in the election process. If all Potential Browsers were to respond at once, things could get a little noisy.[1] So, as with the `AnnouncementRequest` frame, when a browser receives a `RequestElection` frame it will wait a random amount of time before sending its response. The amount of time to wait varies by the status of the node, however. A Potential Browser that is more likely to win the election will send its response to the `RequestElection` frame sooner than one that is less likely.

It's supposed to work like this:

Browser election timings

Response Delay	Node Credentials
0–100 ms	Local and Domain Master Browsers
200–600 ms	Backup Browsers
800–3000 ms	All others

The goal here is to cut down on network broadcast traffic. If the likely candidate votes first, the chances are good that the others won't have to vote at all.

After sending a `RequestElection` frame, a candidate should wait two or three seconds to be sure that all other candidates have voted. After that, if the candidate has won the round it can send another `RequestElection` frame. This marks the start of another round. The election runs four rounds, after which the browser still standing (there should be only one) declares itself the winner by sending a `LocalMasterAnnouncement` frame.

The timings above are provided in the Leach/Naik Browser draft. Whether existing implementations follow these guidelines or not is a question for further study.

1. It is possible that the reason behind this is that some older IP implementations would overflow their buffers if too many UDP packets all arrived at once. There is anecdotal evidence that such a problem did, at one time, exist.

23.1.2 *The Ballot*

The ballot is contained within the `RequestElection` frame which, just to review, looks like this:

```
struct
  {
  uchar  Opcode;
  uchar  Version;
  ulong  Criteria;
  ulong  UpTime;
  ulong  Reserved;
  uchar *ServerName;
  } RequestElection;
```

The `Opcode` and `Reserved` fields can be ignored. The rest comprise the election ballot. The winner of the election is determined by comparing the ballots using a somewhat arcane formula. Here, plain and simple, is how it works:

Test 1

The higher `Version` wins. If they are the same, continue. The only values for `Version` seen on the wire are 0 and 1. Zero is only used when initiating an election by sending a zero-filled election request.

Test 2

Compare the `Criteria`. The higher value wins. If they are equal, continue. The contents of the `Criteria` field still need to be analyzed.

Test 3

The station that has the greatest `UpTime` wins. If they are equal, continue. The `UpTime` is measured in milliseconds,[2] so there is very little chance that two ballots will have the same value.

Test 4

Compare the `ServerName` strings. The first, in comparison order, wins. (E.g. "EARTH" would win over "OIL".)

---

2. The maximum `UpTime` is a little less than 50 days, after which the 32-bit counter will wrap around to zero again.

There is one more test suggested in the Leach/Naik Browser draft. It might be "Test 0" in the list above. Test 0 says, essentially, that a browser that has recently lost an election is still a loser and should remain a loser until several seconds have passed.

Let's rip apart that `Criteria` field, shall we?

The `Criteria` field is handled like an unsigned long integer, but it can also be divided into four subfields, like so:

```
struct
  {
  uchar  OSlevel;
  uchar  BroMajorVers;
  uchar  BroMinorVers;
  uchar  Role;
  } Criteria;
```

The `OSlevel` is the highest order byte and, therefore, has the most impact when `Criteria` values are compared as unsigned longs. There are some known, predefined values, as shown:

`0x01` = Windows for Workgroups and Windows 9x
`0x10` = Windows NT Workstation
`0x14` = Samba default
`0x20` = Windows NT Server

The higher you crank the `OSlevel`, the better your chances of winning an election.

Moving along, the next subfields are the major and minor Browser Protocol Version numbers. In theory, they should have the values 15 and 1, respectively, but Windows 9x systems use 21 and 4 instead.

The final subfield is known as the `Role` field. It is a bitflag field. There seems to be some disagreement regarding the bits, though. Different sources provide different interpretations. The table below provides reasonable approximations.

Browser roles

Bit	Description
`0x80`	Set by the Primary Domain Controller (PDC).
`0x20`	The node is an NBNS client (a P, M, or H node).

Browser roles

Bit	Description
`0x08`	This is the "Preferred Master" bit. It can be enabled manually in Windows via a registry setting, and in Samba by using the `PREFERRED MASTER` option in the `smb.conf` file.
`0x04`	Set by the current Local Master Browser.
`0x02`	Set by a Backup Browser that was until recently the Local Master, but which has been downgraded after losing an election.[3]
`0x01`	Set by Backup Browsers.

It was stated earlier that the LMB election can be rigged so that a specific node always wins. For example, it is necessary that the DMB become the LMB for the LAN.

Higher OS level

In the Windows world, only an NT or W2K server can become a PDC and, therefore, only these can be DMBs. These systems will set the highest defined OS level which, as shown above, is `0x20`. Thus, in a purely Windows environment, the only competition will be from other NT and W2K servers.

Preferred Master

To further bias the LMB election, the "Preferred Master" `Role` bit may be set. This provides an edge over otherwise identical servers. Preferred Master Browsers also force an election whenever they join a LAN.

NBNS Clients

The NBNS client bit is higher order than the Preferred Master bit. Setting this helps because only an NBNS client can contact a remote Domain Master browser to synchronize lists. Thus, an NBNS client is a better choice as an LMB than a B mode node (even a preferred master).

The DMB

The PDC bit is set to ensure that a PDC will win over any other NT or W2K server on the LAN. From a Windows perspective, the PDC must

3. Chances are good that this node is still bitter.

also be the DMB so setting this bit *should* ensure that the DMB will win the Local Master Browser election.

The thing is, there is no guarantee that a third-party browse server will obey the criteria conventions used in Windows. For example, a Samba server can be configured to have an OS level of 255 which would cause it to win the election over the Domain Master. Ouch.

23.2 Timing Is Everything

Several different Microsoft documents provide Browse Service timing information, much of which has already been presented. For the sake of clarity, the Browse Service timings are collected in the table below. These values may be verified against the Microsoft article *Browsing and Windows 95 Networking* as well as the Leach/Naik draft.

Browser Service timings

Period	Operation
15 minutes	**Backup Browser Sync.** The Backup Browser performs a `NetServerEnum2` operation with the Local Master Browser.
15 minutes	**Local Master Browser Sync.** The Domain Master Browser performs a `NetServerEnum2` operation with a Local Master Browser when it receives a `MasterAnnouncement` from the LMB, and then repeats the sync every 15 minutes.
15 minutes	**Domain Master Browser Sync.** Local Master Browsers will contact their Domain Master Browser and perform a `NetServerEnum2` operation to retrieve the merged Browse List.
1 minute, increasing to 12	**Host and Local Master Announcements.** These announcements are sent one minute apart at first. The period typically increases in the following sequence: 1, 2, 4, 8, 12, 12, 12...
1 minute, increasing to 15	**Domain Announcements.** Similar to the previous kind, except that they peg at 15 minutes instead of 12 and the series is reported to be: 1, 1, 1, 1, 1, 15, 15...
36 minutes	The timeout period for a Host entry to time out of the local Browse List. It should be 3 × the announcement period, but in testing, some Providers listed their `Periodicity` incorrectly.

Browser Service timings

Period	Operation
45 minutes	The timeout period for a Domain entry to time out of a foreign workgroup's Browse List.
15/2 minutes	The average amount of time required before a Backup Browser discovers that its Local Master is missing, and calls another election. Elections may also be called if a Preferred Master shows up on the LAN or if a Consumer gets no response to a `GetBackupListRequest`.

If you like playing with numbers (and really, who doesn't) you can spend some time going through the mental exercise of figuring out how long it takes for Host and Domain entries to time out across subnets.

...or you could take a nice quiet walk in the forest. The forest sounds good. Yep. Forest.