# The Air Drop Scenario – Disconnected Networks

## Introduction

This article describes the “air drop” scenario, a method of replicating two Perforce servers that may never be on the same network. For concrete illustration purposes, both server environments are presumed use the Perforce Server Deployment Package (SDP). The SDP implements sophisticated “offline checkpoint” operations, a key detail for implementation.

## Why to Avoid Air Drop (If You Can)

This approach is by no means a best practice from a collaborative development perspective. Collaborative development requires real-time communication, and thus a continuous network connection. However, in environments where security divisions outrank optimum productivity, this “air drop” approach can provide a means of collaboration

The difference in productivity between real-time and air-drop scenarios should not be understated. The difference is far greater than the response time difference of seconds versus a typical once-per-24-hour update cycle might suggest. In cases of complex collaborative problem solving, the discontinuity of human communication between real-time and air drop scenarios can be extreme – months and years instead of minutes and hours.

## A Basic “One Way” Air Drop Scenario

The basic scenario is that two organizations have a need to share deep, rich source-code history. Each company operates its own independent Perforce server. The sharing in this scenario is one-way. We’ll refer to the two sample organizations “Core Technology” (CT) and “Product Development” (PD).

The CT team has the utmost need for security. Source code developed by this team may never been seen outside the physical building where the code is developed. Features such as Disaster Recovery are foregone to achieve the utmost level of security.

The PD team may also have high security requirements, but can allow their code to be shared with the CT team, as is necessary to build a full software product using software from both organizations. The CT team builds software products, which include code from both the CT team and the PD team. (The CT team may extend this approach to a number of PD teams, collectively gathering software from various organizations).

As a means of sharing, each team can, separately and independently, access an intermediate storage system (ISS). The teams have only the most rudimentary access mechanisms to access ISS, such as FTP (but not rsync). The two teams cannot access the storage at the same time. Only a small number of digital assets – a handful of files – can be transferred practically. Further, the ISS has limited storage capacity, significantly less than would be needed to store the full Perforce repositories of either the CT or PD organizations.

## Mechanics of a “One Way” Air Drop

To bootstrap the air drop transfer mechanism, the PD team creates a Perforce checkpoint and a tar file containing all archive files at the point in time represented by the checkpoint. These assets are to be transferred to the CT team. Due to size limitations on the ISS, this one-time bootstrapping process may require a special exception, such as transfer via high-capacity DVD.

The CT team maintains two Perforce servers:

* Server PD-Copy, bootstrapped from the initial checkpoint, is an incrementally updated replica of the PD server made available, read-only, in the CT server environment.   It has rich history, all the checkin comments, jobs data, possibly code review info, etc.
* The CT server that uses the ‘remote depot’ feature to import code file paths representing code drops from the PD-Copy server into the CT server. The remote depot picks up files and file contents (but not the rich history).  Code is imported into a code-drop branch, from where it can be branched and merged into local changes.

Once the CT team has an initial copy of the PD team’s data, routine daily operation can commence. Daily processing works like this:

1. The PD organization modifies its daily checkpoint process – a standard part of routine SDP daily offline backup processing – to touch a timestamp file each day when it starts the checkpoint process.
2. After the checkpoint process, a simple find command, run from the root of the archive file area, detects archive files that changed since the prior day’s timestamp file.  It might look like this:

p4master\_run 1 daily\_backup.sh

# Produces /p4/1/checkpoints/p4\_1.jnl.502

# Customized version also does this:

cd /p4/1/depots

find . -newer archive\_files\_transported -print -exec tar -rpzf daily\_update\_files.502.tgz {} \;

The PD Perforce server is halted during this operation, to ensure archive files are current with the metadata, as suggested by using the journal number in the archive tarfile name.

Because both the metadata and archive files reflect updates of the last 24 hours, the size of these files can fit on the intermediate storage system.

1. The PD team captures the numbered daily Perforce journal file, as well as the similarly numbered tar file containing archive files that correspond to the metadata in the archive file. Both files are transferred to the intermediate shared storage device.
2. The CT team loads those two files, which effectively constitute 24 hours of activity from the PD team.
3. The CT team uses a p4 merge command on their own server, which specifies the remote depot as the source path, and the designated receiving branch in the CT server as the target path. This detects any new code drops from the newly updated PD-Copy server, and makes them ready for builds.

The above procedure repeats daily. Expect that about once annually, re-bootstrapping may be necessary, e.g. to accomplish P4D software upgrades.

## A “Two Way” Air Drop Scenario

To date, no customer we know of has extended the above concept to a two-way solution. More research is needed to determine how feasible this would be.